UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA DE ELETRICIDADE
LABORATÓRIO DE SISTEMAS DIGITAIS

MONTADOR DO "PATINHO FEIO"

Antonio Marcos de Aguirra Massola
João José Neto
Moshe Bain

Julho
1977

Em memória de
Laís Costa Ortenzi

INDICE

Assunto	Página

Assunto	Página

Apêndices | Página

LISTA DE ILUSTRAÇÕES

1 - INTRODUÇÃO

O ante-projeto do minicomputador Patinho Feio nasceu de um curso de pós-graduação dado pelo Professor Glen George Langdon Jr., em 1972. A seguir, os engenheiros e estagiários do Laboratório de Sistemas Digitais (LSD) da EPUSP terminaram o projeto e montaram o Patinho Feio que, dessa forma, se tornou o primeiro computador projetado e construído no Brasil.

Os circuitos do Patinho Feio são totalmente constituídos por circuitos integrados da família TTL ("transistor - transistor logic"), apresentando uma memória de núcleos de ferrite, e tendo um ciclo de máquina de dois microsegundos.

O Patinho Feio foi destinado a pesquisas no LSD, tanto na área de programação ("software") como dos circuitos eletrônicos ("hardware").

Cuidou-se do desenvolvimento de um "software" que permitisse um uso mais eficiente do minicomputador, já que, de início só se podia programá-lo em linguagem de máquina, manualmente, através do seu painel. Em particular, foi definida uma linguagem de montador ("assembly language"), que associa a cada instrução de máquina um mnemônico, e um programa montador ("assembler"), cuja função é traduzir programas escritos em linguagem de montador para linguagem de máquina, os quais são os assuntos tratados neste manual.

Este manual foi escrito de forma a tratar cada tópico de forma mais ou menos extensa, na suposição de que o leitor tenha tido previamente apenas um pequeno contato com a área de computação, e pouco ou nenhum conhecimento de linguagens de baixo nível, como um montador. Por causa disso, tentou-se fazer com que o manual fosse o mais auto-explicativo e independente possível de outros textos. Naturalmente é impossível

que um texto seja completamente independente de outros; por isso, recomenda-se consultar outros textos, tais como manuais de operação do Patinho Feio e de seus equipamentos periféricos (de entrada/saída), textos sobre números binários, etc.

Foi feito um bom esforço para apresentar os conceitos com clareza e para padronizar as notações, com o objetivo de tornar o manual realmente útil. Contudo, certamente muitas falhas subsistem, de forma que são bem recebidas quaisquer sugestões e críticas de modo a melhorar o manual em futuras edições.

Observações:

a) As informações contidas neste manual são as melhores que se pôde obter na época em que o manual foi escrito (setembro de 1975). Contudo, devido ao constante desenvolvimento de novos projetos de "hardware" e "software" para o Patinho Feio, alguns detalhes podem ter sofrido alterações até a presente data.

b) Os programas e trechos de programa existentes no manual foram aí colocados por estarem sintaticamente corretos, mas não representam necessariamente exemplos de boa técnica de programação.

2 - ARITMÉTICA BINÁRIA E HEXADECIMAL
(Com números inteiros)

1. Bases de Numeração

Utiliza-se, na vida diária, a base decimal de numeração para representar os números. Isto significa duas coisas:

a) existem dez algarismos com os quais todos os números são representados (pois a base de numeração é dez), a saber: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

b) emprega-se uma notação posicional onde está subentendido que, quando um algarismo é deslocado de uma posição para a esquerda, seu valor é multiplicado por dez. Por exemplo:
$295 = 2 \times 10^2 + 9 \times 10^1 + 5 \times 10^0$

Generalizando, quando se escreve o número $N = d_n\ d_{n-1} \ldots d_2\ d_1\ d_0$ (sem sinal), onde os d_i $(i = 0, 1, 2, \ldots, n)$ são os seus algarismos (ou dígitos), está-se querendo dizer que: $N = d_n \times 10^n + d_{n-1} \times 10^{n-1} + \ldots + d_2 \times 10^2 + d_1 \times 10^1 + d_0 \times 10^0$.

Nada obriga a que se use apenas a base dez. Na verdade, qualquer base b(inteira) pode ser escolhida para representar um número. Para tanto, escolhem-se b símbolos distintos (os algarismos da base) que representam os números de zero a $(b - 1)$. Escrevendo-se agora $n + 1$ algarismos adjacentes $d_n d_{n-1} \ldots d_1 d_0$ e subentendida a notação posicional descrita acima, tem-se o número N representado por essa notação:

$$N = d_n \cdot b^n + d_{n-1}\ b^{n-1} + \ldots + d_1\ b^1 + d_0\ b^0$$

Inversamente, pode-se provar que cada número N tem uma única representação, numa dada base b, que satisfaz as condições mencionadas acima.

Exemplo: Escolhendo $b = 3$, têm-se três algarismos;

convencionalmente usa-se 0, 1, 2. Então tem-se:

$$(1202)_3 = 1 \times 3^3 + 2 \times 3^2 + 0 \times 3^1 + 2 \times 3^0 = (47)_{10}$$

Pode-se começar a perceber a importância do que foi dito acima quando se considera que os computadores modermos trabalham sempre, em última análise, com a base dois.

2. Bases mais empregadas em computação

Além da base dez, que é de uso geral, empregam-se comumente as seguintes bases:

a) base dois (binária) - necessita dois algarismos distintos para representar os números zero e um. Por convenção utilizam-se os símbolos 0 e 1. Um algarismo binário é também chamado "bit" (do inglês "binary digit").

   A base dois é extremamente importante pois, como já foi citado, os computadores só entendem sequências de zeros e uns , que são usadas tanto para representar as instruções dadas à máquina quanto números prop iamente ditos.

b) base oito (octal) - utiliza os algarismos de 0 a 7. Não será aqui tratada com mais detalhes porque não é utilizada no Patinho Feio, embora o seja em vários outros computadores.

c) base dezesseis (hexadecimal) - os dígitos hexadecimais são: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F; usados para representar os números de zero a quinze.

   Exemplo: $(AB)_{16} = 10 \times 16^1 + 11 \times 16^0 = (171)_{10}$

A correspondência entre os valores binários, decimais e hexadecimais é apresentada na tabela seguinte (note-se que são necessários quatro bits para representar todos os dígitos hexadecimais na base dois).

Decimal	Hexadecimal	Binário
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

3. Conversão de números entre as bases dois, dez e dezesseis

Conforme já se deve ter percebido, surge frequentemente a necessidade de converter números escritos em uma base para outra. Para isso existem algoritmos gerais, dos quais são apresentados abaixo alguns casos particulares:

a) Conversão para a base dez de números escritos em outra base. Basta escrever o número na forma $d_n \cdot b^n + \ldots + d_0$ e efetuar as operações indicadas.

Exemplos: 1º) $(101111100001)_2$ para a base 10

$= 1.2^{11} + 0.2^{10} + \ldots + 0.2^1 + 1 = (3041)_{10}$

2º) $(BE1)_{16}$ para a base 10

$= 11 \times 16^2 + 14 \times 16 + 1 = (3041)_{10}$

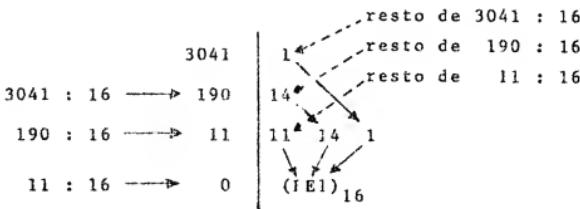
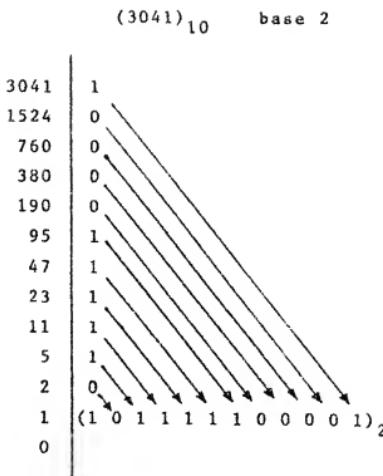
Uma forma conveniente de fazer isso é dada nos diagramas abaixo:

$(BE1)_{16}$ para a base 10

```
                  11    14     1
base              ↓
original=16 ←x─  11   190   (3041)10
```

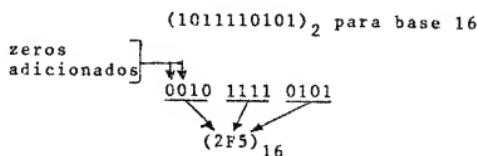
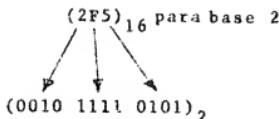
Método usado: $11 \times 16 + 14 = 190$

↓

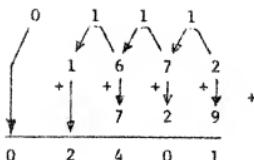
$190 \times 16 + 1 = 3041$

$(10110)_2$ para a base 10

Método usado: $1 \times 2 + 0 = 2$ → $2 \times 2 + 1 = 5$ → $5 \times 2 + 1 = 11$ → $11 \times 2 + 0 = 22$

b) Conversão de números escritos na base dez para uma outra base. Divide-se repetidamente o número dado pela base de destino até que o quociente seja zero. Os restos obtidos são a representação desejada, em ordem invertida. Ver os esquemas abaixo:

$(3041)_{10}$ base 16

```
                        3041 │ 1  ← resto de 3041 : 16
3041 : 16  ──→  190 │ 14 ← resto de  190 : 16
 190 : 16  ──→   11 │ 11 ← resto de   11 : 16
  11 : 16  ──→    0 │ 11  14  1
                     │ (BE1)16
```

c) Conversão entre as bases dois e dezesseis.

c-1) Da base dois para a base dezesseis. Basta agrupar os dígitos binários de quatro em quatro (a partir da direita) e substituí-los pelo respectivo dígito hexadecimal, conforme a tabela apresentada mais atrás (item 2. c).

Exemplo:

c-2) Da base dezesseis para a base dois. Basta substituir cada dígito hexadecimal pelo seu código de quatro bits.

Exemplo:

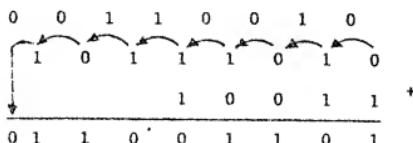
Obs.: 1º) Para a base oito, como é fácil perceber, o método é inteiramente análogo, dividindo-se o número binário em grupos de três bits.

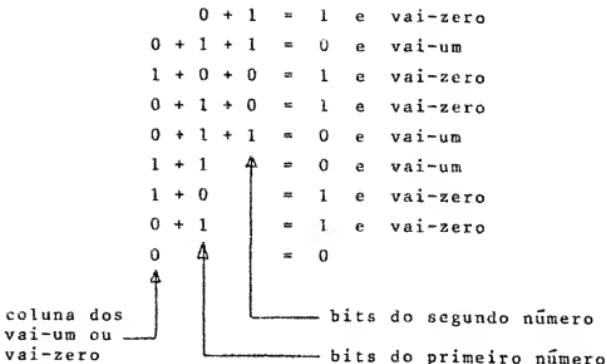
2º) Pode-se agora notar porque são tão usadas as bases oito e dezesseis em computação: elas permitem dividir por três (pois $8 = 2^3$) e por quatro (pois $16 = 2^4$), respectivamente, o comprimento em algarismos do número escrito na base dois, que costuma ser inconvenientemente longo.

3º) Está-se dando mais ênfase à base hexadecimal porque no Patinho Feio os números têm ou oito ou doze bits de comprimento, podendo então, ser representados com dois ou três dígitos hexadecimais, enquanto que, por exemplo, para transformar um número de oito bits (também chamado "byte") em um número octal , tem-se que adicionar um zero à frente do número, para dividí-lo em três grupos de três bits cada.

4. Soma de números binários positivos

Realiza-se de forma inteiramente análoga à soma somum, de números decimais. Para ver isso, examine-se detalhadamente uma soma decimal, por exemplo, de 1672 com 729.

Começando a partir da direita, foram realizadas as seguintes operações:

2 + 9 = 1 e vai-um (= 11)
1 + 7 + 2 = 0 e vai-um (= 10)
1 + 6 + 7 = 4 e vai-um (= 14)
1 + 1 = 2 e vai-zero(= 02)
0 = 0

Com os números binários procede-se da mesma forma, segundo as seguintes regras:

0 + 0 + 0 = 00 ⟶ 0 e vai-um
0 + 0 + 1 = 01 ⟶ 1 e vai-zero
0 + 1 + 1 = 10 ⟶ 0 e vai-um
1 + 1 + 1 = 11 ⟶ 1 e vai-um

Exemplo:

1º) Seja somar 10111010 com 10011. Tem-se:

Começa-se a partir da direita, realizando as seguintes operações:

2º) Somar 11101 com 110.

5. Representação de números negativos

Obs.: Nos itens seguintes assume-se sempre que um número tem oito bits de comprimento, quando for binário.

Até agora, só foram tratados os números positivos . Contudo, é óbvia a necessidade de se manipular números negativos, de modo que é preciso uma representação adequada para os mesmos. Especialmente, é necessária essa representação para números binários, de modo que o computador possa reconhecer os números que sejam negativos como tais.

Existem três modos de representar números negativos em notação binária, chamados de: sinal e amplitude, complemento de um e complemento de dois.

a) Representação de sinal e amplitude.

Usualmente, quando se quer denotar um número como negativo (em qualquer base), coloca-se à sua frente um sinal de menos (-), e quando positivo, às vezes, o sinal de mais (+). Mas, como um computador não reconhece os sinais + e -, mas apenas zeros e uns, vê-se que é necessário reservar um bit do número (geralmente o primeiro) para indicar o seu sinal (usa-se zero para indicar um número positivo e um para indicar um negativo). Supondo um número de oito bits, tem-se, por exemplo:

Desta forma pode-se representar os números inteiros de -127 a +127. Note-se que existem duas representações do número zero, a saber: 0000 0000 e 1000 0000.

b) Representação em complemento de um.

Nesta representação, para indicar um número negativo troca-se os seus zeros por uns e vice-versa. Como sempre, o primeiro bit indicará o sinal do número. Exemplo:

Deste modo, analogamente ao anterior, pode-se representar os números de -127 a +127 e o zero continua com duas representações, a saber: 0000 0000 e 1111 1111.

c) Representação em complemento de dois.

Para se obter a representação em complemento de dois, soma-se um (em binário) à representação em complemento de um, retendo-se apenas os oito bits mais à direita. Exemplo:

A representação em complemento de dois tem as seguintes propriedades:

1a.) O primeiro bit do número indica o seu sinal: positivo se zero e negativo se um.

2a.) São representáveis os números de -128, cuja representação é 1000 0000; a +127, cuja representação é 0111 1111. Desta forma, o número -128 não tem complemento de dois. De fato, tem-se:

-128 ⟶ 1000 0000 ⟶ 0111 1111 +
1
———————
1000 0000 ⟶ +128?

que está errado, pois é a representação de -128, não de +128.

3a.) O número zero tem apenas uma representação: 0000 0000. De fato, tem-se:

0000 0000 ⟶ 1111 1111 +
1
———————
[desprezado ⟶ 1]0000 0000 ⟶ 0000 0000

6. Aritmética no Patinho Feio

Já foi visto como somar números binários positivos e como representar números negativos em oito bits. Deste modo, pode-se passar à soma (e subtração) de números de oito bits, que é o que o Patinho Feio consegue fazer diretamente através de seus circuitos eletrônicos. Ver-se-á também, como operar com números de mais de oito bits e como reconhecer quando o resultado de uma soma não pode ser representado em oito bits (isto é, o número é menor que -128 ou maior que +127).

a) Vai-um:

Denomina-se vai-um de uma soma entre dois números de oito bits ao vai-um na última soma realizada (bit mais significativo), ou seja, ao que seria o nono bit da soma (se fossem considerados números de nove bits). Exemplo:

O vai-um funciona efetivamente como nono bit da soma, permitindo o tratamento de números de qualquer comprimento. Exemplo: seja somar os seguintes números:

a =	0101 1101 1010
b$^+$=	0110 1001 0110$^+$
a+b=	1100 0111 0000

Dividindo a e b em duas partes de oito bits cada uma, vem:

de modo que, com o uso do vai-um, a soma foi realizada corretamente. Note-se que as segundas partes de a e de b não são interpretadas como números negativos de oito bits (apesar de começarem com um), uma vez que é necessário somá-las normalmente (como se fossem números positivos de oito bits, não precedidos de sinal) e obter o vai-um correspondente.

b) Soma e subtração de números no Patinho Feio:

Demonstra-se o seguinte: sejam dois números a e b escritos em notação binária de n bits, com números negativos em notação de complemento de dois. Para calcular a-b, basta somar a com o complemento de dois de b, retendo os n bits menos significativos (desprezando o vai-um). Se a resposta for negativa, estará também em complemento de dois.

Exemplos:

1º) 5 - 6 = -1 $(+5)_{10}$ = 0000 0101 $(+6)_{10}$ = 0000 0110

$(-6)_{10}$ = 1111 1010
0000 0101
vai-um=0 1111 1111 = $(-1)_{10}$

2º) 5 + 6 = 11 0000 0101 = 5_{10}
0000 0110 = 6_{10}
vai-um=0 0000 1011 = 11_{10}

3º) $-5 - 6 = -11$

	1111 1011	=	-5_{10}
	1111 1010	=	-6_{10}
vai-um=1	1111 0101	=	-11_{10}

4º) $-5 + 5 = 0$

	1111 1011	=	-5_{10}
	0000 0101	=	$+5_{10}$
vai-um=1	0000 0000	=	0

Desta forma, consegue-se realizar qualquer soma e subtração, no Patinho Feio, de números de oito bits representados na notação complemento de dois, utilizando-se apenas a soma e a complementação de dois.

Obs.: Convém repetir que nada impede que se considere, se assim for conveniente, a sequência 1111 1010 como um número de oito bits desprovido de sinal (positivo) que valeria então $(250)_{10}$; analogamente 1111 1011 valeria $(251)_{10}$; somando-se estes dois números, obtém-se $(501)_{10}$, que em binário é 1 1111 0101; este é exatamente o resultado obtido na soma, desde que se considere o <u>vai-um</u> como nono bit da mesma.

c) Transbordo:

Como os números foram limitados a oito bits de comprimento, dos quais o primeiro é o bit de sinal, nota-se que existirão valores de <u>a</u> e <u>b</u> tais que, <u>a+b</u> ou <u>a-b</u> seja muito grande ou muito pequeno para ser representado em complemento de dois em oito bits. Contudo, é sempre possível fazer a soma tal como no item anterior, embora ela resulte errada. Neste caso diz-se que houve <u>transbordo</u> (do inglês "overflow").

Exemplo:

$(60)_{10} + (70)_{10} = (130)_{10}$ (o maior número representável é +127)

```
     0011 1100
   + 0100 0110
     ---------
vai-um=0   1000 0010
```

$= (-126)_{10}$ o que está obviamente errado

Conclui-se, portanto, que é necessário ao computador detetar estes casos para evitar erros no processamento. Isto se faz comparando os dois últimos vai-uns ao realizar a soma: pode-se provar que, se eles diferirem entre si, houve transbordo. Exemplo: tomando o exemplo anterior, e detalhando todos os vai-uns, tem-se:

Tome-se agora o seguinte exemplo: 42 - 27 = 15

Pode-se, portanto, realizar qualquer soma e subtração de números de oito bits no Patinho Feio, obtendo-se não apenas o resultado em oito bits na mesma notação de complemento de dois usada nos operandos, como também um vai-um para funcionar como eventual nono bit da soma e uma indicação de transbordo que mostra se o resultado obtido está ou não correto.

7. Blocos e diagramas lógicos

Como serão apresentados mais adiante diagramas lógicos e funções lógicas executadas pelo Patinho Feio, tem-se abaixo, um pequeno resumo:

a) Uma variável lógica é uma variável que pode assumir dois valores, geralmente chamados de 0(zero) e 1(um), ou F(falso) e V(verdadeiro), ou OFF(desligado) e ON(ligado).

b) Uma função lógica associa a cada conjunto de valores de suas variáveis um dos dois valores citados (valor lógico).

Algumas funções são muito utilizadas e têm inclusive representação gráfica como um bloco lógico:

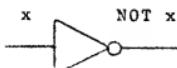
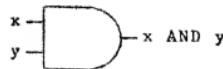
1º) Função NOT (negação) bloco:

x NOT x

Tabela para a função NOT

x	x' = NOT x
0	1
1	0

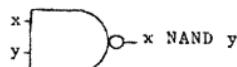
2º) Função AND (e) bloco: x AND y

Tabela

x	y	x . y = x AND y
0	0	0
0	1	0
1	0	0
1	1	1

x . y só é <u>um</u> quando x <u>e</u> y forem ambos <u>um</u>.

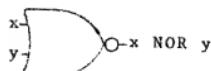
3º) Função OR (ou) bloco:

Tabela

x	y	x + y = x OR y
0	0	0
0	1	1
1	0	1
1	1	1

x + y vale <u>um</u> quando pelo menos uma das variáveis valer <u>um</u>.

4º) Função NAND bloco: x NAND y

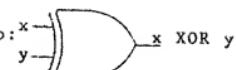
É um AND, seguido de um NOT.

Tabela

x	y	(x.y)' = x NAND y
0	0	1
0	1	1
1	0	1
1	1	0

$(x.y)'$ vale um quando as duas variáveis não forem simultaneamente <u>um</u>.

5º) Função NOR   bloco: x, y → x NOR y

É um OR, seguido de um NOT.

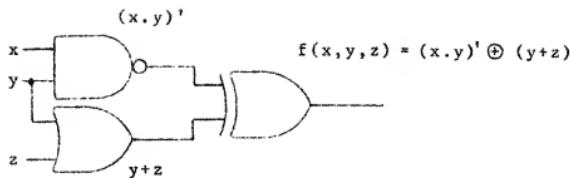



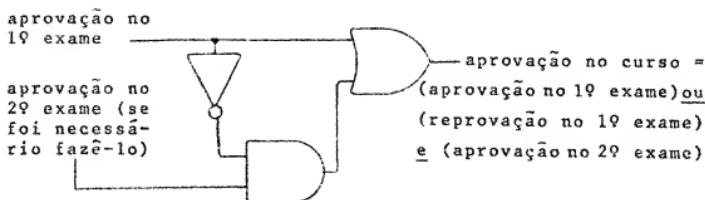
Tabela

x	y	$(x+y)' = x$ NOR y
0	0	1
0	1	0
1	0	0
1	1	0

$(x+y)'$ só vale um se x e y forem ambos zero.

6º) Função XOR (ou exclusivo)   bloco: x, y → x XOR y

Tabela

x	y	$x \oplus y = x$ XOR y
0	0	0
0	1	1
1	0	1
1	1	0

$x \oplus y$ vale um quando x tiver um valor diferente de y.

c) Diagramas lógicos:

Um diagrama lógico é um conjunto de blocos lógicos interligados de forma a constituir uma função lógica.

Exemplo:

Exemplo prático: Suponha-se que, para ser aprovado em um curso, um aluno deve <u>ou</u> ser aprovado no primeiro exame <u>ou</u>, sendo reprovado neste, ser aprovado em um segundo exame. O diagrama lógico correspondente será:

<u>Obs</u>.: 1ª) Não se tratou aqui da multiplicação e divisão de números inteiros binários, nem de operações com números fracionários, pois tais operações não são diretamente realizáveis pelo "hardware" (circuitos eletrônicos) do Patinho Feio.

2ª) As funções lógicas foram tratadas muito superficialmente, pois terão relativamente pouca importância neste manual.

3 - DESCRIÇÃO SUCINTA DO MINICOMPUTADOR "PATINHO FEIO"

Neste capítulo apresenta-se uma pequena descrição da estrutura do computador "Patinho Feio". O objetivo é proporcionar um entendimento mais fácil do montador ("assembler") dessa máquina, através do conhecimento das partes de que esta se constitui. Não é uma explicação detalhada da lógica e funcionamento do computador; para isso deve-se consultar o manual adequado.

Inicialmente, pode-se dividir o "Patinho Feio" em duas partes: o computador propriamente dito (UCP) e a parte que trata de entrada e saída (E/S). A entrada e saída neste computador é discutida nos capítulos 12 e 13.

Do ponto de vista do usuário programador, o Patinho se constitui de cinco registradores, três "flip-flops" (elementos de memória que podem assumir dois estados, designados por 0 e 1 ou "desligado" e "ligado"), uma UCP (Unidade Central de Processamento) e uma memória de núcleos de ferrite, cujo tamanho é 4K (isto é, 4096) palavras de oito bits cada uma(oito dígitos binários).(Também pode-se dizer 4K "bytes", pois um "byte" é um número de oito bits).

As palavras da memória são numeradas de 0 a 4096,ou, em hexadecimal, de /000 a /FFF (os números hexadecimais serão sempre precedidos de /), ou seja, são necessários doze dígitos binários (quatro para cada dígito hexadecimal) para endereçar (isto é, designar a posição de) uma palavra de memória.

Algumas posições da memória têm funções especiais:

a) posição /000, chamada INDEXADOR (IDX), é usada quando há endereçamento indexado (capítulo 4 e 5) e na instrução TRI(capítulo 11).

b) posição /001, chamada "extensão do acumulador" (EXT), é usada na instrução TRE (capítulo 11).

c) posições /002 e /003, são usadas em interrupções, quando guardam o endereço de retorno (capítulo 11).

d) posições de /004 a /F7F, constituem a memória propriamente dita, onde podem ser armazenados livremente programas e dados.

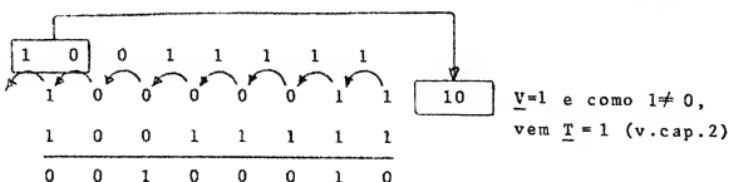
e) posições de /F80 a /FFF, constituem a memória protegida, onde fica guardado o programa carregador ("loader" ou "bootstrap") (capítulo 16). Nesta área nada pode ser armazenado por programas normais em execução.

Os cinco registradores acessíveis por programa são:

a) Acumulador (ACC) - é o principal registrador do Patinho Feio. Tem oito bits de comprimento, portanto, o mesmo tamanho de cada palavra da memória. Todas as operações aritméticas e lógicas usam o conteúdo do acumulador como um dos operandos, e o resultado é aí colocado; todos os desvios condicionais, tipo "pulo", são feitos através de testes no conteúdo do ACC (ver no capítulo 4 os conceitos de "pulo" e "salto"); e todos os dados que entram ou saem da máquina têm que passar pelo ACC.

Exemplo: O Patinho Feio dispõe, por exemplo, das seguintes instruções (explicações mais detalhadas serão vistas em outros capítulos):

CAR carrega (copia) no ACC o conteúdo de uma dada posição de memória.

SOM soma ao conteúdo do ACC o conteúdo de uma dada posição de memória.

PLAN desvia o processamento para uma certa instrução se o conteúdo do ACC for negativo.

SAI saída do conteúdo do acumulador para o meio externo.

b) Registrador de Vai-Um (V) - é um registrador de um bit, que é atualizado, por exemplo, a cada operação aritmética (adição) ou de deslocamento (capítulo 10) realizadas. Após a execução das operações aritméticas, V é o vai-um da operação realizada (vide capítulo 2), isto é, se houve vai-um , então V = 1; caso contrário, V = 0.

c) Registrador de Transbordo (T) - também é um registrador de um bit, que é modificado, por exemplo, cada vez que é realizada uma adição. Se houver transbordo na adição, então é feito T = 1 e isto indica que o resultado(contido no ACC), está errado. Se não houver transbordo, então T = 0.

Nota: quando há transbordo numa operação o processamento não pára; apenas é ligado o bit do registrador correspondente (isto é, T = 1).

Exemplo: a seguinte operação faz com que V = 1 e T = 1.

V=1 e como 1≠ 0, vem T = 1 (v.cap.2)

d) Contador de Instruções (CI) - o CI é um registrador de doze bits (para poder apontar qualquer posição da memória), cujo conteúdo é sempre o endereço na memória onde está a próxima palavra a ser lida e interpretada como uma instrução(ou parte de uma instrução). Quando uma leitura nessas condições acontece, o CI é automaticamente incrementado de uma unidade.
Como se vê, se durante o processamento do programa for mudado o conteúdo do CI, será alterada a execução sequencial das instruções, prosseguindo a execução a partir de uma outra instrução qualquer do programa. Isto,de fato é possível, e as instruções que permitem fazê-lo recebem o nome de instruções de desvio (alteram o conteúdo do CI). Os desvios no Patinho Feio estão divididos em pulos e saltos (v.cap.4).

e) Registrador de Chaves (RC) do painel - o Patinho Feio dispõe, no seu painel, de doze interruptores (chaves) que constituem o registrador de chaves completo, de doze bits. Quando o Patinho Feio não está processando um programa, é possível, através desse registrador, endereçar qualquer posição da memória (por isso são necessários os doze bits), e colocar qualquer valor no CI. Isto feito, pode-se recomeçar a execução do programa (que partirá, portanto, do ponto pré escolhido,ou então pode-se observar e mesmo alterar o conteúdo da posição de memória referida. Para isto, utiliza-se apenas os oito bits mais à direita do RC (RC incompleto),já que o conteúdo de uma posição de memória tem apenas oito bits,
Além disso, pode-se, no meio de um programa, ler o conteúdo do RC incompleto, copiando-o no ACC (capítulo 9). Com isso, podem ser previstas,no programa,intervenções manuais do operador, que alterem a execução.

Obs.: Embora o Patinho Feio tenha outros registradores,eles não são diretamente acessíveis por programa e podem ser considerados como "área de rascunho" para o "hardware" (circuitos do computador).

Os três "flip-flops" existentes no Patinho Feio são os seguintes:

a) "bit" de endereçamento indireto (BEI), usado nas instruções de referência à memória. Quando igual a um, indica que o endereçamento é indireto; quando igual a zero, o endereçamento não é indireto (ver capítulo 5).

b) "flip-flop" que INIBE/PERMITE interrupção do sistema, conforme esteja desligado ou ligado (ver capítulo 11).

c) "flip-flop" que indica se o sistema NÃO ESTÁ/ESTÁ em interrupção (ver capítulo 11).

Uma vez vistos a memória e os registradores de que dispõe o Patinho Feio, passemos a um quadro geral das instruções que ele executa.

Como uma palavra no Patinho Feio tem apenas oito bits, nem todas as instruções puderam ser projetadas para "caber" numa única palavra. Especialmente, as instruções que necessitam do endereço de alguma palavra da memória não podiam caber em oito bits, já que para endereçar toda a memória são necessários doze bits.

Desta forma, foram criadas no Patinho Feio dois tipos de instruções:

a) instruções curtas, que ocupam uma palavra apenas, às vezes, impropriamente chamadas "micro-instruções".

b) instruções longas, que ocupam duas palavras da memória.

No total, o Patinho Feio dispõe de 50 instruções, que foram divididas em grupos, de acordo com suas características. Abaixo temos um quadro com esses grupos e também o(s) capítulo(s) onde elas são tratadas com detalhe:

Além disso, existem comandos para o montador (não são instruções de máquina) chamados pseudo-instruções (capítulos 4 e 15).

Dar-se-á agora uma idéia geral do processamento realizado pelo computador.

Suponha-se então, que a máquina vai começar a processar uma instrução.

Neste instante, o CI tem como conteúdo um número de doze bits, que aponta então para uma palavra da memória. O conteúdo desta palavra é lido e posto numa área de "rascunho". Automaticamente, o CI é incrementado de uma unidade.

A seguir, o Patinho Feio determina, a partir da palavra lida, se a instrução a ser executada é curta ou longa. Se for curta, ela é executada em seguida e passa-se a uma nova instrução (isto é, aquela para a qual o CI estiver apontando).

Se a instrução for longa, é lida na memória a palavra cujo endereço está no CI (como o CI tinha sido incrementado de um, será lida justamente a segunda palavra da instrução longa). Automaticamente soma-se mais uma unidade ao CI.

De posse das duas palavras da instrução longa, pode-se executar a instrução se ela for de E/S, imediata ou deslocamento. Contudo, se for uma instrução de referência à memória, é necessário obter um endereço (chamado "endereço efetivo"). Portanto, inicialmente é calculado um endereço, a partir da instrução a executar (ver capítulo 4), e a seguir a instrução é executada com o endereço assim obtido.

Passa-se então à execução da próxima instrução ("próxima" significa aquela para a qual o CI estiver apontando).

Vê-se, então, que a execução será sequencial, a menos que nossa instrução altere o conteúdo do CI.

As ações acima repetem-se até o computador encontrar uma instrução de parada ou espera (capítulo 11) ou até ser parado manualmente pelo operador.

Nota: Observar que a memória armazena tanto o programa quanto os dados, isto é, uma mesma palavra da memória pode ter seu conteúdo interpretado como instrução (se o CI apontar para ela) ou simplesmente como número binário (por uma outra instrução). Isto permite, inclusive, que no meio de um programa se altere o próprio programa,mudando o conteúdo de alguma palavra que mais tarde venha a ser executada como instrução. Contudo, recomenda-se só usar este recurso em casos especiais,porque ele dificulta a depuração do programa e facilita a ocorrência de erros.

4 - PRINCÍPIOS DO MONTADOR DO PATINHO FEIO

Conforme se pode notar dos primeiros três capítulos, é necessário um certo conhecimento do funcionamento de um computador para entender sua linguagem do montador ("assembler"). Cada computador tem seu montador particular, dependente de suas características de funcionamento.

Mas, o que é o montador? Na verdade, ele nada mais é do que um programa escrito para o computador, cuja função será vista a seguir.

Um computador não entende as linguagens comuns; ele só entende instruções codificadas como números binários. Desta forma, para conseguir que o computador fizesse alguma coisa, o programador teria que traduzir seus pensamentos numa sequência de "zeros" e "uns" compreensível para a máquina e colocar essa sequência na memória. Aí sim a máquina poderia executar as instruções correspondentes. Exemplo: a instrução que manda o Patinho Feio parar um processamento tem o código hexadecimal /9D ou, em binário, 10011101.

Ora, este processo de tradução seria demoradíssimo e sujeito a muitos erros, e além disso, as instruções não são fáceis de lembrar: 10011101, por exemplo, nada significa para a grande maioria das pessoas.

Por isso criou-se o programa montador, que atribui a cada instrução um mnemônico - no exemplo dado, o mnemônico é PARE - e cuja função é justamente tomar um programa escrito com esses mnemônicos e gerar os números binários das instruções de máquina correspondentes, efetuando assim, a tradução , automaticamente. Desta forma o programa fica mais fácil e rápido de corrigir, se houver erros, e mais compreensível.

Há então uma correspondência entre as instruções da máquina e as do montador, isto é, cada instrução da máquina tem o seu mnemônico na linguagem do montador.

Além disso, existem instruções dirigidas especialmente ao programa montador, que permitem controlar o modo como a "tradução" será realizada. São chamadas de "pseudo-instruções", pois não dão origem, normalmente, a instruções que serão executadas mais tarde.

Vejamos quais são as regras gerais a que deve obedecer um programa para ser aceito como válido pelo programa montador.

a) Caracteres:

Os caracteres aceitáveis são os caracteres ASCII, cuja tabela se encontra no apêndice, juntamente com os respectivos códigos binários. Consistem das letras do alfabeto, dos dígitos de 0 a 9 e de caracteres especiais, como ƀ (espaço em branco), =, +, *, (, return , linefeed , etc.

b) Constantes:

O montador do Patinho Feio aceita três tipos de constantes, a saber: hexadecimais, decimais e ASCII; com ou sem sinal.

1º) Uma constante hexadecimal é constituída de uma / seguida por dígitos hexadecimais (0,1,2,...,9,A,B,C,D,E,F) . Exemplo: /A, /10, /398. São permitidos até três dígitos hexadecimais.

2º) Uma constante decimal é constituída de uma sequência de um a quatro dígitos (de 0 a 9).
Exemplo: 0, 025, 1, 118, 2035.

3º) Uma constante ASCII é constituída do caracter @ seguido por um caracter ASCII qualquer.
Exemplo: @1, @A, @*, @ (branco).

No quadro abaixo temos exemplos da representação binária de cada tipo de constante:

Constante	Representação Binária
+10	0000 1010
/10	0001 0000
/5 ou 5	0000 0101
@5	0011 0101
/2E9	0010 1110 1001 (endereço)

Note-se que a diferença de representação entre o número 5(5 como constante decimal ou /5 como hexadecimal) na base 2, que é 0000 0101, e a constante ASCII @5, cuja representação é o código ASCII correspondente ao caracter 5, que é 0011 0101.

Todas as constantes acima podem também ter sinal negativo e neste caso sua representação binária é o complemento de dois da constante positiva correspondente.
Exemplo:

Constante	Representação
-5	1111 1011
-/10	1111 0000
-@5	1100 1011

Exemplos de constantes ilegais: -1+; 1,5; 2.75; /XYZ; @PQ, etc.

Obs.: Quando é fornecida ao montador uma constante demasiadamente grande, apenas os "bits" menos significativos são considerados.
Exemplo: Supondo que se deva colocar, em uma palavra da memória do Patinho Feio, uma constante, apenas os oito bits da direita serão considerados.

Constante Fornecida	Representação Binária	Constante Armazenada na Memória
1	0000 0001	0000 0001
4096	1 0000 0000 0000	0000 0000
-@5	1100 1011	1100 1011
+459	1 1100 1011	1100 1011

c) Símbolos:

Um símbolo é uma sequência de uma a sete letras do alfabeto, das quais são reconhecidas apenas as duas primeiras e a última - três caracteres, portanto. Se o símbolo contiver menos que três letras, o espaço em branco é preenchido pelo montador com o caracter @ .

Exemplo:

PTX
PTAX   representam o mesmo símbolo interno PTX
PTABX

A   representa o símbolo interno A@@

UV   representa o símbolo interno UV@

F99
9B5
AB+   são símbolos ilegais
?2/

Obs.: 1º) O montador admite um máximo de 256 símbolos em cada programa (ou em cada unidade de um programa, se for o montador relocável - capítulo 14).

2º) Nomes de registradores, instruções e posições especiais de memória, como ACC, IDX, V, T, etc., podem também ser usados como símbolos, embora recomende-se não fazê-lo para evitar confusões. Nesta

apostila, os nomes dos citados registradores e posições especiais são sempre sublinhados, para evitar ambiguidade.

d) Rótulos ("Labels"):

Um rótulo é um símbolo ou um "." (ponto) que funciona como nome de uma posição de memória. Portanto, para se referenciar essa posição de memória, pode-se tanto dar seu endereço de doze bits como seu nome de três caracteres, ou ainda a localização do . (ponto) no programa. Exemplos serão vistos depois.

Obs.: Um máximo de 256 posições podem ser rotuladas com um ponto, em cada programa.

e) Mnemônicos:

Mnemônicos são nomes dados às instruções de máquina e são empregados de uma forma fixa, isto é, já estão definidos na própria linguagem do montador. Exemplo: o mnemônico PARE . Na verdade, pode-se alterar o mnemônico, contanto que se mantenham os dois primeiros e o último caracter, já que só esses vão ser reconhecidos pelo montador. Exemplo: PARE poderia ser transformado em PAE, PARTE, PANOTE, etc. Isto não é, de modo algum, recomendado, por razões de facilidade de entendimento.

f) Comentários:

Um comentário é uma sequência de caracteres quaisquer, exceto return e linefeed , que são ignorados no processo de montagem, sendo apenas copiados na listagem do programa (se esta existir). Serve para documentar o programa, isto é, facilitar o entendimento através de uma explicação do que está

acontecendo e do que o programa está fazendo.Comentários devem, portanto, ser usados extensamente nos programas.

Estrutura de um programa escrito na linguagem do montador:

A seguinte convenção será seguida de agora em diante: para indicar conteúdo de alguma coisa, esta será colocada entre os sinais "{" e "}" . Exemplos:

a) - endereço /92F indica a posição de memória /92F.
- {/92F } = /25 indica que o conteúdo da palavra referenciada acima, de endereço /92F, é /25. Note-se que o endereço de uma palavra tem doze bits, mas seu conteúdo é de apenas oito bits.

b) Se agora essa posição de memória for rotulada com o nome EXEM, ter-se-á, na notação empregada:
EXEM = /92F (o endereço)
{EXEM}= /25 (o conteúdo)

c) Podemos ter ACC = 0 (conteúdo do acumulador)
{IDX } = 92 (conteúdo do indexador) , etc.

Obs.: Apenas para os registradores V e T escreveremos V = 1 em vez de {V}= 1, etc., uma vez que T e V não são posições de memória, não havendo assim ambiguidade.
Pela mesma razão serão utilizados os símbolos {return} e {linefeed} para os caracteres especiais ASCII correspondentes. ACC, CI e RC também não são posições de memória, e portanto não serão escritos entre chaves(ver exemplo acima).

Todo programa em linguagem do montador deverá ter a seguinte estrutura:

Um programa é composto de linhas (perfuradas em fita de papel, por exemplo), no seguinte formato:

1º) A primeira linha começa no primeiro caracter da fita que não for um "feed-frame" (nenhuma perfuração na fita).

2º) O fim de uma linha é indicado pela sequência de caracteres {return} e {linefeed}, nessa ordem.

3º) A primeira linha de um programa tem que ser uma linha de controle para o montador (ver capítulo 16).

4º) Todas as outras linhas têm que ter o formato descrito mais adiante.

5º) A segunda linha tem que ser a declaração do tipo ou origem do programa (uma primeira explicação encontra-se mais adiante neste capítulo (pseudo-instruções); uma discussão mais avançada está no capítulo 15).

6º) A última linha tem que ser uma declaração FIM (mesma observação do item anterior). Naturalmente, para terminar esta linha, é necessário {return} e {linefeed}.

Formatos de uma Linha:

Uma linha pode ser linha de comentário ou de instrução, e cada um dos tipos tem o seu formato particular.

a) uma linha de comentário tem o seguinte formato:

```
*  <comentário>
↑
na coluna 1 ─┐
             ↓
Exemplo:     **+COMENTE SEUS PROGRAMAS {return} {linefeed}
```

Obs.: Os símbolos " < " e " > " indicam um elemento que deve ser fornecido pelo programador (no caso, o comentário). Comentários podem ser postos em qualquer parte do programa, entre a 2ª e a última linhas.

b) uma linha de instrução é dividida em campos, de acordo com o seguinte formato:

< rótulo > b̸ < mnemônico > b̸ < operando > b̸ < comentário >

Para separar os campos usa-se ao menos um espaço em branco.

O campo do rótulo é opcional. Se existir, deve começar obrigatoriamente na coluna 1 da linha, e deve conter um símbolo ou um . (ponto), que designarão, daí por diante, o endereço correspondente à linha em que aparecem, conforme exemplo mais adiante.

O campo do mnemônico é obrigatório e deve conter o mnemônico da operação a executar.

Obs.: Se não existir o rótulo, então a coluna 1 dever conter um branco para indicar ao montador que "acabou o rótulo", isto é, que o que vem a seguir é mnemônico e não rótulo.

O campo do operando depende da instrução. Existem instruções que não precisam de operando, e neste caso este campo não existe. Já outras instruções necessitam de operando, e neste caso, o conteúdo deste campo tem o formato que a instrução exigir. (serão explicados conforme forem sendo apresentadas as instruções).

O campo do comentário vai deste ponto até o fim da linha, e serve para comentar as instruções sem necessidade de usar uma linha inteira para este fim. Pode-se colocar neste

campo quaisquer caracteres, exceto naturalmente {return} e {linefeed} , que terminam a linha.

Obs.: 1) Não confundir espaço em branco (ƀ), cujo código ASCII é /20, com "feed-frame" (nenhuma perfuração na fita), de código /00 e ao qual não corresponde nenhuma imagem gráfica.

2) Quando for cometido um erro na perfuração de uma fita, pode-se furar a seguir:

1º) A^c (A controlado) - apaga o último caracter perfurado.

ou

2º) {rubout} ou DEL - apaga a linha inteira até e inclusive o próximo {return} {linefeed}

Ex.: 1º) *ISTO E UM COMIA^{c}ENTARIO {return} {linefeed}

2º) *IXTO E UM COM{rubout}{return}{linefeed}

Pseudo-Instruções:

Serão vistas agora as pseudo-instruções para o montador absoluto (ver capítulo 15, para as pseudo-instruções relocáveis). Tratam-se de instruções para o montador propriamente dito, que podem alterar um pouco o modo como certas partes do programa a ser montado são encaradas pelo montador. Ver-se-á alguns exemplos, que servirão também para esclarecer o que foi explicado antes.

1) ORG (origem):

Define a origem de um trecho do programa, ou seja, a posição a partir da qual ele deve ser armazenado. Tem operando,

que é o endereço da referida posição e deve ser uma constante (costuma-se usar só hexadecimal). Esta instrução não pode ter rótulo.

Exemplo:

```
┌──coluna 1
↓
b ORG b /92A
ALO b PARE
```

Com o acima, a instrução PARE(/9D)ficará armazenada na posição /92A da memória, que foi rotulada com o nome ALO, ou seja: ALO = /92A (endereço)
{ALO} = /9D (conteúdo)

Como foi dito anteriormente, a pseudo-instrução ORG deve estar contida na segunda linha do programa. Contudo, nada impede que haja outras ORG no programa, que especificarão novas origens a partir das quais as instruções seguintes serão armazenadas sequencialmente (até achar outra ORG). Nestas outras ORG o operando pode ser qualquer referência à memória (ver capítulo 5), desde que previamente definida.

2) DEFC (define constante):

Coloca na palavra "corrente" da memória o dado especificado no operando, que deve ser uma constante de qualquer tipo.

Exemplo:

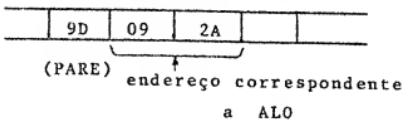
```
    ORG  /92A
ALO PARE          Com isso,  LET   = /92B
LET DEFC -@B                {LET} = /BE (= -@B)
```

Obs.: Como pode ser visto do exemplo acima, existe implicitamente um apontador (posição"corrente"da memória) , cuja posição inicial é cada na pseudo ORG e que vai sendo incrementada a cada linha,conforme o número de palavras ocupadas pela instrução ou pseudo-instrução

dessa linha (por exemplo, a instrução curta PARE,ocupou uma palavra; a pseudo DEFC também ocupou uma palavra onde colocou a constante /BE).

3) BLOC:

Reserva na memória uma área de dados cujo comprimento em palavras é dado no operando, que deve ser uma constante. Não é armazenado nada nessa área, ela é apenas reservada.
Exemplo:

```
      ORG   /101
UPT   BLOC  10      Reserva 10 palavras, nos  endereços
ALO   PARE          compreendidos entre /101 e /10A
```

Tem-se, então, por exemplo, UPT = /101; ALO = /10B.

4) DEFE (define endereço):

Usa <u>duas</u> palavras da memória (a corrente e a próxima) onde é colocado o endereço calculado a partir do operando, com o 1º dígito hexadecimal da 1ª palavra feito igual a zero.
Exemplo:

```
      ORG   /92A     A memória ficará com a seguinte con
ALO   PARE           figuração:
MKA   DEFE  ALO
XTU   ....
```

	ALO	MKA		XTU
	/92A	/92B	/92C	/92D
	9D	09	2A	
	(PARE)	endereço correspondente a ALO		

```
ALO     = /92A
MKA     = /92B
{MKA}   = /09
{MKA+1}= /2A   ,  etc.
```

O operando deve ser do mesmo formato que aquela para as instruções de referência à memória (ver mais adiante).

5) DEFI (define indireto):

Análogo ao anterior, mas o 1º dígito hexadecimal da 1a. palavra é feito igual a um ao invés de zero.
Exemplo:

```
        ORG   /92A
ALO     PARE
MKA     DEFI  ALO
XTU     ....
```

	ALO	MKA		XTU
	/92A	/92B	/92C	/92D
	9D	19	2A	
	(PARE)			

```
MKA   = /19
MKA+1 = /2A
```

As instruções DEFE e DEFI são usadas quando há endereçamento indireto (ver capítulo 5).

6) EQU (equivalência):

Esta instrução serve para dar nomes diferentes à mesma posição de memória.
O rótulo é obrigatório e é o nome adicional da posição de memória cujo endereço é calculado a partir do operando. O operando deve ser do mesmo tipo que aqueles de instruções de referência à memória (ver mais adiante), com a restrição de que o endereço correspondente deve ser possível de calcular imediatamente ao surgir a instrução. Portanto, tudo o que aparece no operando deve estar previamente definido.

Exemplo:

```
APT   EQU   /722     faz com que os símbolos APT e QTX
QTX   EQU   /722     referenciem a mesma posição de me-
                     mória de endereço /722.
```

Outro modo de conseguir a mesma coisa é:

```
APT   EQU   /722
QTX   EQU   APT
```

Mas está errado escrever:

```
QTX   EQU   APT
APT   EQU   /722 , pois no primeiro EQU,o operando APT
                   ainda não está definido.
```

7) FIM:

Serve para terminar a montagem e deve ser a última instrução do programa (o programa só pode ter uma pseudo-instrução FIM). O operando também deve ser do mesmo tipo que o de uma instrução de referência à memória, e indica o endereço onde deve ser iniciada a execução do programa. Este operando tem apenas o objetivo de documentar o programa.

Desvios no Patinho Feio

Como já dissemos, as instruções são executadas sequencialmente, a menos que se altere o valor do CI, efetuando assim um desvio para uma outra posição do programa.

Existem dois tipos de desvios no Patinho Feio:

a) Pulos:

Um pulo ocorre quando é colocado no CI um valor pré-determinado pelo programador e à sua livre escolha. Desta forma, consegue-se pular para qualquer posição arbitrária da memória.

b) Saltos:

Um salto ocorre quando o Patinho Feio soma duas unidades ao conteúdo do CI, desta forma saltando duas palavras (o espaço para colocar uma instrução longa).

Conforme será visto, existem instruções que permitem realizar pulos, enquanto outras poderão resultar apenas em saltos.

5 - INSTRUÇÕES DE REFERÊNCIA À MEMÓRIA ENDEREÇAMENTO NO PATINHO FEIO

Em toda instrução de referência à memória é necessário (como o próprio nome indica) obter o endereço de uma posição de memória (chamado endereço efetivo), antes de executá-la. Existem, no Patinho Feio, 4 modos de obter esse endereço:

1º) Endereçamento direto
2º) Endereçamento indexado
3º) Endereçamento indireto
4º) Endereçamento indireto indexado

Todas as instruções de referência à memória são longas e, portanto, têm um comprimento de 4 dígitos hexadecimais. Desses, o 1º dígito é o código da instrução, isto é, indica a operação que deve ser executada. Os outros três dígitos são usados para o cálculo do endereço efetivo.

Vamos denotar o endereço efetivo por eee, e os 3 últimos dígitos da instrução por nnn.

1º) Endereçamento direto:

Neste caso, é feito eee = nnn , isto é, o endereço efetivo é diretamente aquele especificado na instrução.

2º) Endereçamento indexado:

Neste caso, ao endereço nnn especificado na instrução, é somado o conteúdo do indexador (posição /000 da memória) , para a obtenção do endereço efetivo, isto é, eee = nnn + {IDX}.

Exemplo: se nnn = /220 e {IDX} = /1F, então eee = /23F.

3º) Endereçamento indireto:

O Patinho Feio possui um "bit" chamado BEI (bit de endereçamento indireto) que, quando está ligado, indica que o endereçamento é indireto. Este bit é ligado pela instrução IND, e é desligado automaticamente após a execução de qualquer instrução que não seja IND.
No endereçamento indireto, nnn aponta uma posição de memória. Ao invés de tomar o conteúdo desta posição como dado para executar a instrução, o Patinho Feio toma o conteúdo desta e da próxima palavras como um novo endereço, onde será achado o dado necessário, a menos que o quarto bit do conteúdo da primeira palavra tomada seja 1, pois neste caso, o conteúdo do novo endereço seria não o dado, mas um novo endereço, e assim por diante, até que seja encontrada uma palavra cujo quarto bit seja 0. Este será finalmente o endereço efetivo do operando.

Exemplo:

1º) seja nnn = /125, com endereçamento indireto

seja /125 = /17
/126 = /F2 → 1¦7F2

então o novo endereço é /7F2 e o endereçamento continua indireto por causa do 1 (pois /1 = 0001_2 e o quarto bit é um).

Sendo agora /7F2 = /00
/7F3 = /26 → 0¦026

então o novo endereço é /026 e o endereçamento não é mais indireto.

Portanto, o endereço efetivo é /026 e o dado usado na execução da instrução é { /026 } (o conteúdo de /026).

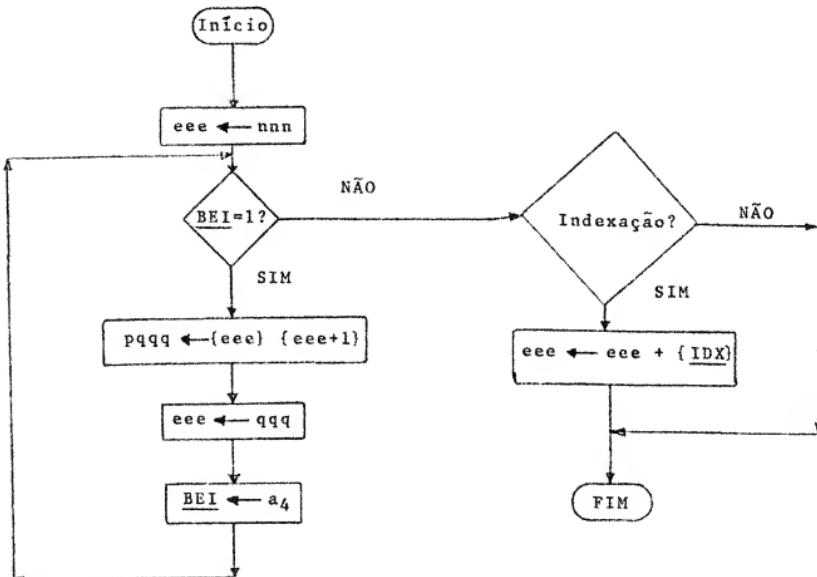
2º) seja nnn = /125, com endereçamento indireto

seja /125 = /11 \
/126 = /25 / ——→ 1¦125

o novo endereço é ainda /125 e o endereçamento continua indireto.

Vê-se que o Patinho Feio nunca vai acabar de calcular o endereço efetivo pois, sempre estará procurando novo endereço nas mesmas posições de memória, e o endereçamento indireto nunca acaba. Chama-se a isso "loop de endereçamento indireto".

Apertando-se o botão "endereçamento" do painel, o Patinho Feio pára a execução do programa após terminar a instrução que estiver executando. Mas no caso de "loop" de indireto, isto não serve, pois a instrução não termina nunca de ser executada. Então, o único modo de parar o Patinho Feio, nesse caso, é apertar o botão "preparação".

Obs.: Deve-se evitar apertar o botão "preparação" quando o Patinho Feio não estiver parado, pois poderá resultar na destruição do conteúdo da memória. Para parar um processamento, deve-se apertar o botão "endereçamento" (exceto, naturalmente, quando houver "loop" de indireto).

4º) Endereçamento indireto indexado:

É a união dos dois tipos de endereçamento. Uma vez calculado o endereço final resultante do endereçamento indireto, a ele é somado o conteúdo do indexador. É o que se chama "pós-indexação".

Diagrama de Blocos:

O diagrama de blocos abaixo explica como é feito o cálculo do endereço efetivo pelo Patinho Feio.

Representando: o endereço efetivo por eee
os últimos três dígitos hexadecimais da instrução por nnn
o conteúdo de duas palavras adjacentes da memória por pqqq (dígitos hexadecimais)
os "bits" de p por $a_1a_2a_3a_4$
BEI = bit de endereçamento indireto(BEI=1 ⟹ endereçamento indireto)
IDX = indexador (posição /000 de memória)

Operandos de Instruções de Referência à Memória:

Estes operandos são usados pelo montador para calcular a parte nnn do código de máquina .

Os operandos podem ser os seguintes:

1) <símbolo> - referencia o endereço da instrução ou do dado onde o símbolo foi usado como rótulo.
(endereço simbólico puro)

2) <símbolo> <sinal> <deslocamento> - referencia o endereço do caso anterior somado ou subtraído ao deslocamento (em número de palavras). O deslocamento é uma constante da qual são considerados apenas os últimos quatro bits, dando, portanto, um deslocamento máximo de quinze palavras.
(endereço simbólico relativo)

3) <endereço> - é uma constante, cujos últimos doze bits, convertidos para notação hexadecimal, constituem o próprio nnn a calcular.
(endereço absoluto)

4) * - é o endereço da 1ª palavra da própria instrução. Exemplo: o modo mais fácil de conseguir um "loop" de indireto é com a pseudo-instrução DEFI * .
(endereço relativo puro)

5) * <sinal> <deslocamento> - análogo ao anterior, só que deslocado para cima ou para baixo, conforme o <sinal> .
(endereço relativo puro)

6) *-* - o mesmo que /000.
(endereço absoluto)

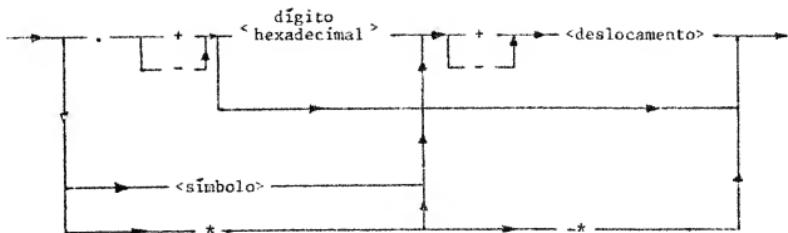
7) . <sinal$_1$>- refere-se ao endereço da instrução rotulada com um . mais próxima para cima (.-) ou para baixo (.+) da instrução atual.
(endereço local puro)

8) .<sinal$_1$> <dígito hexadecimal N (sem / antes)> - refere-se ao endereço da instrução distante de N pontos para cima ou para baixo da instrução atual. Se N for omitido, será considerado por omissão igual a 1.
(endereço local puro)

9) .<sinal$_1$> <N> <sinal$_2$> <deslocamento> - análogo ao anterior, só que ainda com deslocamento para cima ou para baixo (de acordo com o <sinal$_2$>). Se N for omitido, então é considerado por omissão igual a 1.
(endereço local relativo)

Exemplos:

1º) Suponhamos que APT = /120 e que a instrução atual vai ser armazenada em /300. Então temos:

operando	nnn resultante
APT	/120
APT + 15	/12F
/722	/722
*	/300
*-1	/2FF
-	/000

2º) <u>endereço</u> ORG /100

```
         endereço   ORG  /100
           100     . DEFC  5
           101     . DEFC  6
  ┌──→ 102 e 103   <mnemônico><operando1>
  │        104     . PARE
  │                  FIM  <operando2>
  │
  └── instrução  longa
```

operando	número do operando	nnn resultante	
. -	1	/101	
. - 2	1	/100	
. +	1	/104	
. -	2	/104	
. - 2	2	/101	
. - 3	2	/100	
. - 3 - 1	2	/0FF	Obs.:Cai fora do programa!
. - 3 + 2	2	/103	
. + + 1	1	/105	
. - - 3	1	/0FF	Obs.:Cai fora do programa!

<u>Diagrama de Precedências para os operandos das instruções de referência à memória:</u>

O diagrama seguinte resume tudo o que foi dito sobre os operandos aceitáveis nas instruções de referência à memória. Para construir um operando válido, pode-se seguir qualquer caminho no diagrama seguinte, de acordo com o sentido indicado pelas flechas. Os elementos contidos entre os caracteres "<" e ">" devem ser fornecidos pelo programador, enquanto que os outros devem aparecer na mesma forma e posição que no diagrama a seguir mostrado.

Serão vistas agora as instruções propriamente ditas que, conforme já foi dito, são todas longas. Além disso, será vista, também, a instrução curta IND, que apesar de não ser uma instrução de referência à memória, é sempre usada em conjunto com estas.

Instrução IND (indireto) - código de máquina /9F - operando: não tem.
Liga o bit (BEI) que indica endereçamento indireto. Este bit é desligado pela próxima instrução, qualquer que seja (exceto, naturalmente, outra IND). Portanto, se a próxima instrução não referenciar a memória e, portanto, não exigir o cálculo de um endereço, a instrução IND não terá nenhum efeito e terá sido desperdiçada.

Instruções de Referência à Memória:

Instrução PLA (pula) - código de máquina Onnn.
Pula incondicionalmente (isto é, independentemente do valor de qualquer registrador) para o endereço eee (endereço efetivo), calculado a partir de nnn, sem indexação.

Isto se consegue colocando eee no contador de instruções, pois este sempre indica o endereço da próxima instrução a ser executada. Ou seja, o Patinho Feio, quando durante um processamento encontra uma instrução Onnn, calcula eee e faz CI ← eee. A seguir, vai executar a próxima instrução, cujo endereço está no CI, ou seja, eee, como era desejado.

Exemplo:

```
         ORG   /100
         PLA   CSI
    CSI  PLA   XUN
    XUN  PLA   *       (é um loop)
         FIM
```

Instrução PLAX (pulo indexado) - código de máquina 1nnn.

Esta instrução é análoga à anterior, exceto que o endereçamento é indexado. Ocasiona um pulo incondicional para eee, calculado a partir de nnn com indexação.

Exemplo:

Supondo PLAX /207 e { (IDX) = /2B ; BEI = 0 }   eee = /207 + /2B = /232

CI ← /232   pula para a instrução de endereço /232.

Instrução ARM (armazena) - código de máquina 2nnn.

Copia o conteúdo do acumulador na posição de memória de endereço eee ((eee) ← ACC), calculado a partir de nnn nem uso do indexador. O conteúdo anterior da posição eee da memória é perdido, e o conteúdo do acumulador não se altera.

Note-se que, embora esta instrução seja longa ,

ela altera apenas uma palavra da memória, já que o ACC é um registrador de oito bits de comprimento.

Exemplo:

```
      ARM    .+         Guarda o conteúdo do ACC na
       :                primeira palavra da instru-
    • PLA    ...        ção PLA.
```

Instrução ARMX (armazena indexado) - código de máquina 3nnn.

Analogamente à anterior, faz {eee} ← ACC, mas agora, no cálculo de eee a partir de nnn. usa-se também o indexador.

Exemplo:

```
ARMX  *   ┌{IDX} = /01                  /207  /208  /209
          │ BEI  = 0        antes        32    07
          < ACC  = /00
          │ Sup.*= /207     depois       32    00
          └
```

Explicação:

```
   nnn  = /207 +
  {IDX} = / 01
  ------------
   eee  = /208
```

Vai armazenar ACC = /00 na 2ª palavra da própria instrução (modifica a instrução!).

Instrução CAR (carrega) - código de máquina 4nnn.

Copia o conteúdo de eee no ACC (ACC ←{eee}), sem modificar o conteúdo de eee;o conteúdo anterior do ACC é perdido; eee é calculado de nnn, sem o uso do indexador.

Instrução CARX (carrega indexado) - código de máquina 5nnn.
Análogo à instrução CAR, mas no cálculo de eee a partir de nnn, é usada também a indexação.

Instrução SOM (soma) - código de máquina 6nnn.
Calculado eee a partir de nnn, sem indexação, soma os conteúdos do ACC e eee e coloca o resultado no ACC (ACC ← ACC + {eee}), perdendo seu conteúdo anterior. eee não se altera. Os bits V (vai-um) e T (transbordo) são ligados ou desligados, conforme tenha ou não havido vai-um e transbordo nesta soma (independentemente do seu valor anterior). (ver aritmética binária no Patinho Feio, capítulo 2).

Exemplo:

```
        :
        CAR   UM
        SOM   DOIS
        ARM   TRES
        :
```

faz {TRES} = {UM} + {DOIS}

```
se    {UM}   =  0111 1100
      {DOIS} =  0101 1001
                ---------
vem   {TRES} =  1101 0101 , V=0, T=1 (houve trans
      ACC    =  1101 0101   bordo e vai-um).
```

Instrução SOMX (soma indexado) - código de máquina 7nnn.
Análogo a SOM, faz ACC ← ACC + {eee} e atualiza V e T. A diferença é que no cálculo de eee é usada a indexação.

Instrução PLAN (pula se negativo) - código de máquina Annn.

Se ACC < 0 (isto é, se seu bit mais à esquerda for 1), pula para a instrução de endereço eee (o que se consegue fazendo CI ← eee), onde eee é o endereço efetivo, calculado a partir de nnn. Não é possível, nesta instrução, o uso do indexador para este cálculo. Caso contrário (se ACC > 0) segue sequencialmente.

Instrução PLAZ (pula se zero) - código de máquina Bnnn.

Análoga à instrução PLAN, mas o pulo se dá somente se ACC = /00. Ou seja, se ACC = /00 , então CI ← eee. Aqui também não é possível o uso da indexação no cálculo de eee.

Exemplo: Programa que multiplica A por B, somando A+A+...+A; B vezes.

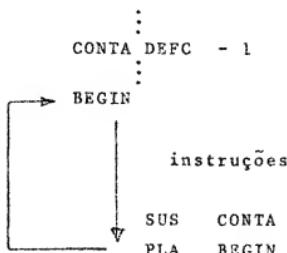
```
        ORG     /372
A       DEFC    <valor de A>
B       DEFC    <valor de B>
P       DEFC     0        lugar para armazenar o produto
MEN     DEFC    -1        número menos um
.       CAR      B        TESTA SE B É ZERO
        PLAZ     .+       PULA PARA .+ E PÁRA SE FOR
        SOM      MEN'     SUBTRAI UM DE B SE NÃO FOR
        ARM      B        E GUARDA EM B.
        CAR      P        SOMA NOVAMENTE O NÚMERO
        SOM      A        A AO PRODUTO P.
        ARM      P
        PLA      .-       E VOLTA A TESTAR B.
.       PARE     *
        FIM      .-2
```

Instrução SUS (subtrai um ou salta) - código de máquina Ennn.
O Patinho Feio calcula eee a partir de nnn (sem o uso do indexador). A seguir testa o conteúdo de eee.

Se {eee}= 0 salta duas palavras (ou seja, soma 2 ao CI. (Lembre-se da diferença entre pulo e salto - ver capítulo 4),

Se {eee}≠ 0 subtrai um do conteúdo de eee (ou seja, {eee}←{eee} - 1).

Esta instrução é a única que pode fazer uma operação aritmética diretamente na memória(subtrair 1 de {eee}), sem usar o acumulador e não alterando nenhum dos outros registradores (portanto, V (vai-um) e T (transbordo) não se alteram, mesmo se houver transbordo e/ou vai-um na operação de subtração).

O principal uso desta instrução é para controlar a execução de um grupo de instruções que devem ser repetidas um número pré-determinado de vezes. Também pode ser usada para contar quantas vezes um certo trecho de programa foi executado. Vide os exemplos a seguir.

Como Controlador:

```
NUM    DEFC     nº de vezes a executar
         :
CONTR  SUS   NUM  (comentário 1)
       PLA   EXEC
         :
         :     instruções a executar após o
         :     processamento repetitivo.
EXEC
         :
         :     instruções a executar repetitivamente.
         :
       PLA   CONTR
```

Comentário 1:

Enquanto {NUM} ≠ 0, faz {NUM}←{NUM} - 1 e executa a próxima instrução - PLA EXEC - portanto, pula para EXEC e processa as instruções até PLA CONTR, quando volta para testar mais uma vez o valor de {NUM}. Finalmente, quando for {NUM} = 0 salta as duas palavras seguintes que contêm a instrução PLA EXEC (pois PLA é instrução longa). Portanto, não executa PLA EXEC e continua a execução sequencialmente.

Como Contador:

Comentário:

Como CONTA = -1 < 0, já da primeira vez que o SUS é executado, é fácil ver que o salto não será executado e sempre haverá a volta ao BEGIN. Cada vez que o grupo de instruções for executado subtrair-se-á um de CONTA. Portanto, fica-se com:

Nº de vezes executado	CONTA
0	-1
1	-2
2	-3
⋮	⋮
n	-n-1

Ou seja, há a contagem de quantas vezes foram executadas as instruções (com sinal trocado).

Obs.:

Devido ao fato de a aritmética do Patinho Feio ser com complemento de 2, deve-se tomar cuidado para não usar este método de contagem para um número muito grande de execuções, pois pode ocorrer o seguinte:

nº de execuções	valor de CONTA	
125	-126	
126	-127	
127	-128	perde-se o valor do nº de vezes.
128	+127	
129	+126	
⋮	⋮	
253	+2	
254	+1	
255	0	haverá um salto sobre a instrução PLA BEGIN e a sequência de instruções não tornará a ser executada.

Explicação:

$$
\begin{array}{rcl}
1000\ 0000 & = & -128_{10} \\
+\ 1111\ 1111 & = & -\ \ 1_{10} \\
\hline
0111\ 1111 & = & +127_{10}
\end{array}
$$

V = 1   T = 1   mas como SUS não mexe em V e T, não ficaremos sabendo que houve transbordo.

Exemplos:

1) Somar os elementos de uma matriz de 100 elementos armazenados a partir da posição MAT, isto é:

```
{MAT }    =  a1
{MAT+1}   =  a2
   :
{MAT+99}=  a100
```

supondo que {IDX} = 100

```
              :
SOMA    DEFC  0
              :
CONTR   SUS   0
        PLA   EXEC
              :
EXEC    CARX  MAT
        SOM   SOMA
        ARM   SOMA
        PLA   CONTR
              :
```

2) O programa já descrito para multiplicar A por B, pode ser refeito com a instrução SUS.

```
     ORG    /372
A    DEFC   valor de A
B    DEFC   valor de B
P    DEFC   0       LOCAL P/ARMAZENAR O PRODUTO.
.    SUS    B       TESTA SE B JÁ É ZERO;
     PLA    *+3     SE NÃO FOR,VAI P/A INSTR.CAR P;
     PARE           SE FOR, PÁRA .
     CAR    P       SOMA MAIS UMA VEZ
     SOM    A       A AO PRODUTO JÁ
     ARM    P       ACUMULADO.
     PLA    .-      VOLTA PARA TESTAR B
     FIM    .-
```

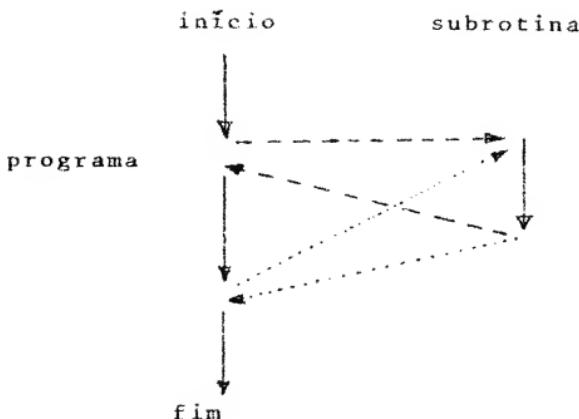
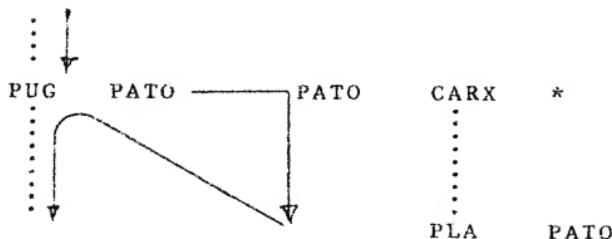
Notar o uso do operando *+3 :

- * refere-se à primeira palavra da instrução PLA.
- *+1 refere-se à segunda palavra da instrução PLA.
- *+2 refere-se à instrução curta PARE.
- *+3 refere-se à primeira palavra da instrução CAR.

Instrução PUG (pula e guarda) - código de máquina Fnnn.

Descrição da execução da instrução pelo Patinho Feio:

1º) Calcula o endereço efetivo eee a partir de nnn. Não usa indexação.

2º) Seja CI = /klm (endereço da próxima instrução que seria executada):

Faz {eee} = /0k
[eee+1]= /1m
CI = eee+2

Explicação da instrução:

Esta instrução serve para a implementação de subrotinas no programa, ou seja, um grupo de instruções que deve ser executado a partir de vários pontos do programa, aos quais se deve voltar após uma execução,conforme o diagrama exemplo:

Note-se como esta instrução consegue isto:

Supondo que o endereço correspondente a PATO é /202 e que a instrução PUG ocupa as posições de memória /105 e /106 (é instrução longa!), a execução dessa instrução, passo a passo, dá-se da seguinte forma:

	104	105	106	107		201	202	203	204	
memória ...		F2	02		... ...		52	02		...
		PUG /202					CARX *			

Inicialmente:

CI = /105 Lê, na posição indicada pelo CI , /105 da memória, a instrução /F2 e soma 1 ao CI .

CI = /106 Identifica a instrução como sendo longa.

CI = /106 Lê, na posição indicada pelo CI , /106 da memória, a 2ª metade da instrução longa: /02 e soma 1 ao CI (portanto, a instrução é /F202).

CI = /107 Calcula o endereço efetivo:eee=/202.

CI = /107 Se CI = klm, então k = /1, l = /0, m = /7. Coloca em eee = /202 os dígitos hexadecimais /0k = /01, e em eee+1 = /203 coloca /lm = /07.

CI = /107 Faz CI = eee + 2 = /204

CI = /204 Fim da execução da instrução.

A instrução que será executada em seguida é, evidentemente, aquela no endereço /204, pois este é o conteúdo do CI . Contudo, veja-se o que aconteceu nas posições /202 e /203 que foram alteradas pela execução da instrução PUG. Seu conteúdo agora é o seguinte:

	201	202	203	204	
...		01	07		...

Mas, /0107 nada mais é que uma instrução PLA /107, e a posição /10' é aquela imediatamente seguinte à instrução PUG.

O Patinho Feio executa agora toda a rotina, a partir da posição /204, até chegar à instrução PLA PATO, quando há o desvio para a posição PATO = /202. Aí encontra-se uma instrução de pulo de volta para o local de onde havia sido "chamada" a subrotina e, portanto, há a volta ao ponto desejado.

Está claro, então, o que faz a instrução PUG <endereço> : pula para a posição < endereço>+2 e guarda o endereço de chamada em < endereço > e<endereço>+1. Note-se que o conteúdo anterior do <endereço> e <endereço>+1 (no caso a instrução CARX *) simplesmente não interessa, pois o Patinho Feio monta aí, durante a execução, a instrução de retorno, destruindo o que havia antes. A última instrução da subrotina deve ser um PLA <endereço> pois aí se encontra o endereço para onde se deve retornar (local de chamada).

6 - INSTRUÇÕES IMEDIATAS

(Exceto instruções de deslocamento - cap. 10)

São constituídas de quatro instruções, todas elas longas. São chamadas imediatas porque, embora realizando operações aritméticas ou lógicas, não necessitam do cálculo de um endereço efetivo (eee) da memória: o dado sobre o qual se vai operar já vem na própria instrução. Em todas estas instruções, o <operando> é convertido em nn (2ª palavra da instrução). O operando, então, deve ser uma constante, não podendo ser uma referência à memória.

Instrução XOR - código de máquina D2nn.

Realiza a função lógica XOR ("Exclusive Or"), bit a bit, entre o ACC e nn; o resultado fica no ACC. XOR não mexe em V nem em T. Lembrando a tabela de função XOR, tem-se os seguintes exemplos:

Exemplo 1: Instrução: XOR @M

ACC = 1100 1010
@M = 0100 1101
novo ACC = 1000 0111

XOR

a	b	a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0

Exemplo 2: Uma maneira (há outra melhor - vide capítulo 7) de achar o complemento de 1 do ACC, isto é, trocar os zeros por uns e os uns por zeros, é executar um XOR do ACC com /FF:

ACC = 1011 0100
XOR /FF   /FF = 1111 1111
novo ACC = 0100 1011

Instrução NAND - código de máquina D4nn.

Realiza a função lógica NAND, bit a bit, entre ACC e nn (segunda palavra da instrução), colocando o resultado no ACC. Não altera V nem T.

Exemplo 1:

	ACC	=	0100 1101
NAND 25	25_{10}	=	0001 1001
	novo ACC	=	1111 0110

NAND

a	b	$\overline{a.b}$
0	0	1
0	1	1
1	0	1
1	1	0

Exemplo 2: Uma instrução NAND /FF terá exatamente o mesmo efeito que um XOR /FF (ver na instrução anterior): acha o complemento de 1 do acumulador.

Exemplo 3: Quer-se saber se o bit menos significativo é zero ou um. Pode-se fazer um NAND com /01:

	ACC	=	0010 0111	0010 0110
NAND /01	/01	=	0000 0001	0000 0001
	novo ACC	=	1111 1110	1111 1111
			se era um, o resultado é /FE	se era zero, o resultado é /FF

A seguir, pode-se testar o acumulador e desviar conforme o resultado.

Instrução SOMI (soma imediata) - código de máquina D8nn.

Realiza a soma: ACC ← ACC + nn; acerta V e T, conforme o resultado dessa operação.

Exemplo 1:

	ACC	=	0110 0011
SOMI 25	25_{10}	=	0001 1001 +
novo	ACC	=	0111 1100 ; V = 0 , T = 0

Exemplo 2:

	ACC	=	1000 0010
SOMI -33	-33_{10}	=	1101 1111 +
novo	ACC	=	0110 0001 ; V = 1 , T = 1(houve transbordo)

Instrução CARI (carrega imediato) - código de máquina DAnn.

Coloca nn no ACC, sem mexer em V nem em T.

ACC ← nn.

Exemplo: Um modo de zerar o ACC é executar a instrução CARI 0. (Há um modo melhor, ver capítulo 7).

7 - INSTRUÇÕES CURTAS DO GRUPO 1

São todas instruções curtas e, por isso, são às vezes chamadas, impropriamente, de micro-instruções. São do grupo 1 as instruções curtas cujo código de máquina começa com /8. Nenhuma das instruções deste capítulo tem operando.

Instrução LIMPO - código de máquina /80.

Limpa o ACC, V e T.

(Obs.: o último caracter do mnemônico da instrução é o número zero, não a letra "O").

ACC ← 0
V ← 0
T ← 0

Instrução UM - código de máquina /81.

Faz o ACC igual a um e limpa V e T.

ACC ← 1
V ← 0
T ← 0

Instrução CMP1 - código de máquina /82.

Complementa de 1 o ACC e limpa V e T.

ACC ← complemento de um do ACC .
V ← 0
T ← 0

Instrução CMP2 - código de máquina /83.

Complementa de 2 o ACC e atualiza V e T conforme o resultado dessa operação.

Como se sabe, o complemento de 2 é obtido a partir do complemento de 1, somando-se-lhe o número 1, e no Patinho Feio, o complemento de 2 de um número é esse número com sinal trocado.

ACC ← complemento de 2 do ACC .
V e T atualizados segundo o resultado da soma de 1 ao complemento de um do número.

Exemplo 1:

ACC = 0110 1011
CMP2
novo ACC = 1001 0101 , V = 0 , T = 0

Exemplo 2:

ACC = 1000 0000
CMP2
novo ACC = 1000 0000 , V = 1 , T = 1

Instrução LIM - código de máquina /84.

Limpa V e T.

V ← 0
T ← 0

Instrução INC (incrementa) - código de máquina /85.

Soma um ao ACC e atualiza V e T, conforme o resultado da operação.

ACC ← ACC + 1 ; atualiza V e T.

Exemplo: As duas sequências a seguir, têm exatamente o mesmo efeito:

```
 :
CMP1          :
        e   CMP2
INC           :
 :
```

<u>Instrução UNEG</u> - código de máquina /86.

Coloca -1 no <u>ACC</u> e limpa <u>V</u> e <u>T</u>.

<u>ACC</u> ← -1
<u>V</u> ← 0
<u>T</u> ← 0

<u>Instrução LIMP1</u> - código de máquina /87.

Limpa o <u>ACC</u> e <u>T</u>; faz <u>V</u> = 1.

<u>ACC</u> ← 0
<u>V</u> ← 1
<u>T</u> ← 0

8 - INSTRUÇÕES CURTAS DO GRUPO 2

Todas estas instruções ocupam uma só palavra, e seu código de máquina começa com /9. Todas elas podem resultar em saltos, dependendo das condições encontradas. Convém lembrar que um salto é sempre sobre duas palavras, e que em duas palavras cabe uma instrução longa ou duas curtas. O operando destas instruções é uma constante - 0 ou 1, apenas.

Cód.Máq. (hexad.)	Instrução	Operando	Faz CI ← CI +2 se: (Resumo)	Descrição Salta duas palavras quando:
90	ST	0	Se T=0	T=0
91	STM	0	Se T=0, T← 1	T=0,e também faz T←1
92	ST	1	Se T=1	T=1
93	STM	1	Se T=1, T← 0	T=1,e também faz T←0
94	SV	0	Se V=0	V=0
95	SVM	0	Se V=0,e faz V← 1	V=0, e também faz V←1
96	SV	1	Se V=1	V=1
97	SVM	1	Se V=1,e faz V← 0	V=1,e também faz V←0

Exemplo: O trecho de programa abaixo calcula quantas vezes houve vai-um ao se somar seguidamente um ao nº -128 antes de haver transbordo; o resultado fica em CSI.

```
        LIMPO          Limpa a posição CSI
        ARM     CSI
ROT     CAR     KLA    (Supor que KLA contém o nº -128,no início)
        INC            Soma um
        ST      0      Não havendo transbordo continuamos
        PLA     FTP    Havendo, pulamos p/FTP(fim do trecho de programa)
        ARM     KLA    Salva valor atual de KLA
```

```
        SV     1      Se V for zero,vamos para o próximo valor
        PLA    ROT
        CAR    CSI    Se V for 1, somamos um a CSI
        INC           e vamos para o próximo valor.
        ARM    CSI
        PLA    ROT
FTP
```

Note-se que após as instruções ST 0 e SV 1 há as instruções <u>longas</u> PLA FTP e PLA ROT, que ocupam as <u>duas palavras</u> que serão saltadas se as condições forem satisfeitas.

9 - INSTRUÇÕES DE PAINEL

Fazem parte do grupo 1 de instruções curtas, isto é, seu código de máquina começa com /8 e ocupam oito bits.(As outras instruções deste grupo estão no capítulo 7).

Consistem de 8 instruções com o mesmo mnemônico PNL. Apenas o operando muda: é uma constante de zero a sete. Os respectivos códigos de máquina vão de /88 a /8F.

As instruções de painel usam os 8 bits menos significativos (bits à direita) do registrador de chaves (RC) do Patinho Feio (ver capítulo 3). (Os 4 bits mais significativos são ignorados). Quando, neste capítulo 9, utilizamos a abreviação RC, entenda-se como 8 bits, e não como o registrador de chaves completo, que tem 12 bits.

Descrição das Instruções:

Mnemônico	Operando	Cód.Máq. (hexad.)	Descrição	
PNL	0	88	ACC ← RC ; V ← 0 ; T ← 0	
PNL	1	89	ACC ← RC + 1	atualizam V e T conforme o resultado da operação.
PNL	2	8A	ACC ← RC - ACC -1	
PNL	3	8B	ACC ← RC - ACC	
PNL	4	8C	ACC ← RC + ACC	
PNL	5	8D	ACC ← RC + ACC +1	
PNL	6	8E	ACC ← RC - 1	
PNL	7	8F	ACC ← RC ; V ← 1 ; T ← 0	

É claro, então, porque só são usados os 8 bits menos significativos do registrador de chaves: eles deverão ser colocados no ACC ou somados a este, e o ACC só tem 8 bits.

Das instruções mencionadas, praticamente só se usa PNL 0, e eventualmente PNL 7. As outras instruções, normalmente não se usam.

Se os valores de V e T forem necessários, recomenda-se não usar as instruções PNL 1 a PNL 6, porque devido ao "hardware" da máquina (isto é, os seus circuitos eletrônicos), as regras de atualização de V e T podem não ser aquelas que, à primeira vista, parecem verdadeiras (principalmente as instruções PNL 2 e PNL 5).

Normalmente utiliza-se estas instruções para permitir que o operador altere a execução de um programa, entrando com diferentes valores no RC.

É necessário, então, que o computador esteja parado ou em espera (ver capítulo 11), antes de executar uma instrução de PNL, para dar tempo ao operador de introduzir seu dado no RC (após o que ele aperta o botão de PARTIDA ou, às vezes, INTERRUPÇÃO (ver capítulo 11)).

Ao executar a instrução PNL o Patinho Feio não espera o operador colocar o dado no RC; executa a instrução com o valor de RC naquele instante.

Exemplo: o seguinte techo de programa aceita um dado do painel de controle.

```
     :
     PARE        O operador coloca o dado e aperta partida.
     PNL   0     Transfere o dado do RC p/o ACC
     ARM   *-2   e daí para a posição *-2 da memória.
     :
```

Nota: A posição *-2 é onde está armazenada a instru ção PARE, que, portanto, será destruída ao ser executado o ARM. Portanto, se esse trecho de programa for executado novamente, o Patinho Feio não pára antes de executar o PNL, a não ser que o dado, na primeira vez, tenha sido /9D, que é o código de máquina da instrução PARE.

10 - INSTRUÇÕES DE DESLOCAMENTO

Estas instruções fazem parte do grupo de instruções imediatas (capítulo 6); portanto, são todas longas. Seu operando é uma constante entre 0 e 4.(*)

A primeira palavra do código de máquina destas instruções todas é /D1. A segunda palavra é da forma /mn, onde m diz qual é a operação de deslocamento a realizar e n é o operando.

Todas estas instruções operam exclusivamente com o ACC e o registrador V.

O quadro seguinte mostra esquematicamente a ação do computador ao executar cada instrução, com <operando> = 1. Se o operando for n , executa-se o anterior n vezes. Se o operando for zero, não executa nada (deslocamento de zero posições).

(*) Na realidade, o número de deslocamentos é dado pelo número de bits ligados do hexadecimal n da instrução. Exemplo: se n = 0110 ou n = 0011 ou n = 1001, etc., haverá em todos esses casos, dois deslocamentos. Contudo, o operando n deve valer de 0 a 4, e os códigos de máquina realmente gerados são:

operando = 0 → n = 0000
operando = 1 → n = 0001
operando = 2 → n = 0011
operando = 3 → n = 0111
operando = 4 → n = 1111

MNEMÔNICO	NOME	CÓD.MÁQ. (HEXAD.)	ESQUEMA(Com <operando> = 1)	COMENTÁRIOS
DD	Deslocamento à direita	D1 0n	0 → a→a→a→a (ACC) a→a→a→a → V → perdido	zeros entram à esquerda
GD	Giro à direita	D1 2n	a→a→a→a → a→a→a→a → v → perdido (circular)	
DDV	Deslocamento à direita c/V	D1 1n	a→a→a→a → a→a→a→a → v (circular)	as duas instruções são equivalentes
GDV	Giro à direita com V	D1 3n		
DE	Deslocamento à esquerda	D1 4n	perdido v ← a←a←a←a ← a←a←a←a ← 0	zeros entram à direita
GE	Giro à esquerda	D1 6n	perdido v ← a←a←a←a ← a←a←a←a (circular)	
DEV	Deslocamento à esquerda c/V	D1 5n	v ← a←a←a←a ← a←a←a←a (circular)	as duas instruções são equivalentes
GEV	Giro à esquerda com V	D1 7n		
DDS	Deslocamento à direita com duplicação de sinal	D1 8n	s→a→a→a → a→a→a→a → V → perdido (s duplicado)	o bit de sinal(s) é duplicado

Empregamos a notação V = v, ACC = aaaa aaaa ou ACC = saaa aaaa, quando queremos destacar o bit de sinal. Evidentemente, os a representam dígitos binários e não são todos iguais em geral.

Exemplos de deslocamentos: ACC = 1011 0101 V = 0

Se executarmos só a instrução	Teremos o resultado ACC	V	
DD 2	0010 1101	0	
GDV 3	0101 0110	1	
GE 4	0101 1011	1	
DDS 1	1101 1010	1	
DEV 0	1011 0101	0	(não faz nada)

Exemplo: GE 4

	v	a a a a	a a a a
situação inicial	0	1 0 1 1	0 1 0 1
um giro	1	0 1 1 0	1 0 1 1
giro 2	0	1 1 0 1	0 1 1 0
giro 3	1	1 0 1 0	1 1 0 1
giro 4	1	0 1 0 1	1 0 1 1

Resultado final: ACC = 0101 1011

V = 1

conforme está na tabela exemplo.

Nota: A instrução DDS é usada para divisão inteira por 2 (isto é, se o número for ímpar, subtrai-se 0,5 do resultado da divisão (arredondamento)).

Exemplo:

$\underline{ACC} = 0011\ 0101 = 53_{10}$

DDS 1 $\rightarrow$ $\underline{ACC} = 0001\ 1010 = 26_{10} = \frac{53}{2} - 0{,}5$

$\underline{ACC} = 1011\ 0101 = -75_{10}$

DDS 1 $\rightarrow$ $1101\ 1010 = -38_{10} = -\frac{75}{2} - 0{,}5$

$\underline{ACC} = 0100\ 1100 = 76_{10}$

DDS 1 $\rightarrow$ $0010\ 0110 = 38_{10} = \frac{76}{2}$ (o número é par, por isso, não há arredondamento)

11 - CONCEITO DE INTERRUPÇÃO
INSTRUÇÕES CURTAS DO GRUPO 3

Introduzir-se-á neste capítulo, o importante conceito de interrupção e, além disso, serão vistas as instruções curtas do grupo 3, a maioria das quais referentes a esse assunto.

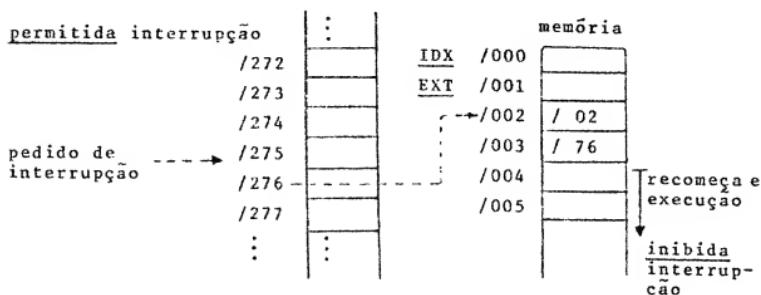
Inicialmente, é necessário dizer que há dois modos de conseguir uma interrupção no Patinho Feio: 1) por meio de um equipamento de entrada e saída, e 2) por meio do botão interrupção do painel. Neste capítulo só será discutido o 2º modo. No capítulo 12 tratar-se-á do outro (E/S). O Patinho Feio só tem um nível de interrupção (adiante isto será explicado com detalhes).

Considere-se, então, a sequência de execução das instruções pelo Patinho Feio. Uma vez que ele comece a executar instruções, ele prossegue sequencialmente, executando-as uma a uma, exceto quando encontra uma instrução de desvio ("pulo" ou "salto"), que altera a ordem normal de execução. E assim, ele prossegue, até encontrar uma instrução PARE. Note-se que não é possível o operador intervir nessa sequência, a menos que ele pare o computador manualmente (ver capítulo 16 - "Operação").

Suponhamos agora, que o operador perceba, de algum modo, que seu programa tem um pequeno erro não-previsto (isto é, que só se tornou visível durante a execução), e suponhamos que, em outra parte da memória, haja uma rotina que lhe permita recuperar-se desse erro. Seria ótimo, então, que se pudesse interromper a sequência normal de processamento e executar essa rotina de correção de erros. Mas como fazê-lo?

Um método óbvio se oferece: parar manualmente o computador e recomeçar a execução a partir da primeira instrução da rotina em questão (ver capítulo 16). Acabando essa rotina, parar novamente a máquina e recomeçar o programa a partir do ponto em que se tinha parado originalmente. Como é fácil ver, esse método, por envolver muitas operações manuais, é lento e está sujeito a erros - por exemplo, é necessário lembrar do ponto onde se havia parado.

Para que tudo isso seja feito automaticamente,criou-se a interrupção, que permite alterar a sequência de execução de instruções pelo lado de fora da máquina.

O nome interrupção (do inglês "interrupt") confunde um pouco: parece que há interrupção no processamento, ou seja, que a máquina pára. Isto não é verdade: a interrupção não pára o computador; ela interrompe a sequência normal de execução das instruções.

Isto é feito do seguinte modo: quando o botão de INTERRUPÇÃO é pressionado, o computador recebe um pedido de interrupção. Ele termina de executar a presente instrução e, então, verifica se a interrupção está permitida ou inibida. Suponhamos que esteja permitida. Neste caso, nas posições 2 e 3 da memória, é montada automaticamente a instrução PLA CI , ou seja, um pulo para a instrução que seria executada a seguir, se não houvesse interrupção, e começa a executar as instruções a partir da posição 4 (faz CI ← 4). Novas interrupções ficam , por enquanto, inibidas.

Percebe-se então que foi, na realidade, executada uma instrução PUG 2 (veja capítulo 5), como se ela estivesse inserida entre a instrução após a qual a interrupção ocorreu e a instrução seguinte.

Nas posições 2 e 3 tem-se, agora, o endereço de onde se deve recomeçar o programa interrompido, aí posto automaticamente pelo computador. A "outra parte da memória", referida alguns parágrafos acima, onde está a rotina de correção de erros, começa na posição 4 da memória.

A grande vantagem do método acima é que a instrução PUG 2 não estava originalmente no programa, mas o computador foi forçado a executá-la, apertando-se o botão de interrupção.

Exemplo:

Em geral, a primeira coisa que se deve fazer ao começar a rotina de interrupção, é guardar o conteúdo do ACC, do IDX e da EXT e, eventualmente, de V e de T, em posições de memória conhecidas. Isto porque, como provavelmente a rotina vai utilizar e modificar essas posições e registradores, seus valores anteriores, necessários ao prosseguimento do programa, seriam destruídos. Essa operação é chamada "salvar o estado da máquina no instante da interrupção".

Logo antes de recomeçar o processamento do programa, deve-se recolocar os valores originais nos respectivos lugares, para que o programa recomece exatamente no estado em que parou.

A isto se chama "restaurar o estado da máquina após o atendimento da interrupção" (isto é, restaurar os valores existentes no momento em que houve a interrupção).

Foi dito mais atrás que o Patinho Feio tem apenas um nível de interrupção. Isto quer dizer que, uma vez que foi aceito um pedido de interrupção, e o Patinho Feio esteja então processando uma interrução, novas interrupções estão inibidas. Ou seja, não adianta ficar apertando o botão INTERRUPÇÃO do painel: o Patinho Feio não aceitará novas interrupções enquanto não acabar de processar a atual.

A razão disto é simples: se fosse aceita mais uma interrupção, o novo endereço de retorno destruiria o antigo ao ser guardado também nas posições 2 e 3 da memória. Em consequência, não se saberia mais onde ocorreu a primeira interrupção, o que evidentemente não deve acontecer.

Como foi executada, no instante da interrupção, uma instrução PUG 2, poder-se-ia pensar em acabar a rotina com um PLA 2. Deste modo haveria a volta à posição 2 e ali seria encontrada a instrução de retorno ao ponto em que se havia abandonado o programa.

De fato, este método funciona se não se quiser, depois, novas interrupções. Mas se for encarada a eventual necessidade de se ter mais interrupções, não é possível acabar assim a rotina, pois novas interrupções estão inibidas! É necessária uma instrução especial que será vista a seguir.

Instruções Curtas do Grupo 3:

Agora, então, serão vistas as instruções relacionadas com interrupções, as quais são um subconjunto do grupo 3 de instruções curtas, que é composto de 8 instruções, todas curtas. Os códigos de máquina correspondentes vão de /98 para a primeira, a /9F para a última. Nenhuma tem operando!

Instrução PUL - código de máquina /98
Pula para a posição de memória /002 e limpa o estado de interrupção. Esta é a instrução necessária para terminar uma interrupção, conforme a discussão anterior. É equivalente a um PLA 2 e termina a atual interrupção; por isso, uma nova interrupção fica agora permitida.
Esta instrução é o único modo de, durante um processamento, terminar uma interrupção. Quando o Patinho Feio está parado, pode-se também apertar o botão "preparação" (ver cap. 16).

Instrução TRE - código de máquina /99
Troca o conteúdo do acumulador com o da extensão (posição /001). Isto permite acesso rápido ao conteúdo da extensão sem que a informação contida no ACC seja perdida.

$$\underline{ACC} \leftrightarrows \{\underline{EXT}\}$$

Instrução INIB - código de máquina /9A
Inibe a interrupção do computador.
Explicação:
Pode acontecer de existir um programa ou um trecho de programa em que não se quer que haja interrupção de modo algum, mesmo que alguém, acidentalmente, aperte o botão INTERRUPÇÃO.

Para estes casos existe esta instrução. Após executado um INIB, não é mais possível interromper o sistema até a execução da instrução PERM (descrita a seguir). Esta instrução não tem efeito se o computador estiver tratando uma interrupção, pois neste caso a interrupção já está inibida.

Instrução PERM - código de máquina /9B

Permite interrupção do sistema.

Esta instrução serve para tornar a interrupção novamente permitida, uma vez que ela esteja inibida pela execução da instrução INIB. Caso contrário, ela não terá efeito algum.

Atenção: Se o Patinho Feio já estiver em interrupção, esta instrução não vai tornar nova interrupção permitida. O único modo de fazer isso é terminar a atual interrupção através de uma instrução PUL.

Nota: Além da interrupção pelo painel, estas instruções também inibem e permitem interrupções de equipamentos de E/S. (Ver capítulo 12, para mais detalhes).

Exemplo:

Instrução ESP (espera) - código de máquina /9C

Pára o processamento até acontecer um pedido de interrupção ou ser acionado o botão de partida. Esta instrução é mais usada no caso de interrupções por periféricos de E/S (capítulo 12).

Instrução PARE - código de máquina /9D

Pára o processamento, que só recomeça quando for acionado o botão de partida. A diferença, então, entre as instruções PARE e ESP é que, ao ser executado um PARE, não adianta apertar o botão de interrupção que o processamento não recomeça, o que acontece no caso da instrução ESP.

Instrução TRI - código de máquina /9E

Troca o conteúdo do acumulador com o do indexador (posição 0 da memória). Isto permite acesso rápido ao conteúdo do indexador sem perder a informação contida no ACC.

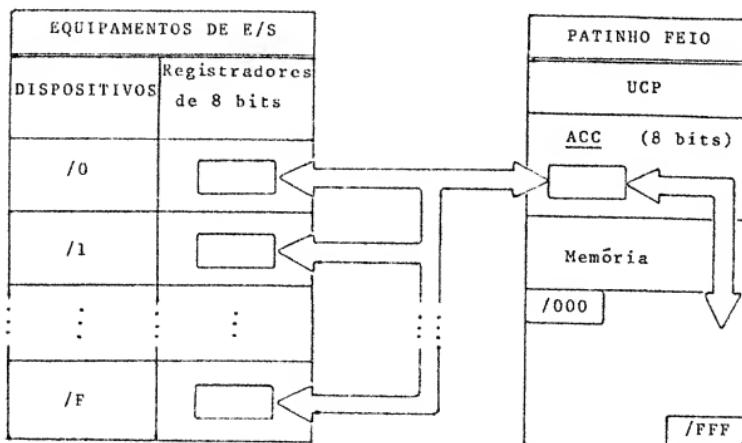
ACC ⇆ (IDX)

Instrução IND - código de máquina /9F

Indica que na próxima instrução o endereçamento é indireto. Se a instrução não for de referência à memória, o IND não terá efeito algum. O uso desta instrução e os modos de endereçamento estão discutidos com detalhes no capítulo 5.

Ver-se-á agora um pequeno exemplo de programa que usa interrupção. Não é um exemplo típico, porque não é necessário salvar o estado da máquina no momento da interrupção e a entrada e saída de dados são manuais.

O programa aceita dados pelo registrador de chaves e acumula numa somatória. Após colocado o dado, o Patinho Feio entra em espera e pode-se apertar o botão de partida para somar o novo dado ao resultado anterior da somatória. Apertando-se o botão de interrupção, o novo dado será subtraído do resultado anterior.

```
        ORG    /4
        CMP2        Troca o sinal do número e volta
        PUL
        ORG    /173
        LIMPO
        TRE         Limpa ACC  e {EXT}
        LIMPO
LER     PARE
                    P/colocar o dado (o resultado da somatória
        PNL    0    pode ser lido no ACC)
        ESP         Para decidir de soma ou subtrai
        SOM    1     ACC <- ACC  + {EXT}
        ARM    1    Guarda novo valor da somatória na EXT
        PLA    LER
        FIM    /173   Execução começa em /173
```

12 - MÉTODOS DE ENTRADA E SAÍDA DE DADOS

Neste capítulo será iniciado o estudo de uma das partes mais importantes (para qualquer computador), ou seja, a entrada e saída de dados, que é o modo de trocar informações entre a máquina e o mundo exterior. Há diferenças muito grandes entre os métodos e instruções de E/S (entrada e saída), de máquina para máquina, e por isso, este assunto é um dos mais difíceis de aprender, na programação de qualquer computador.

Será vista, naturalmente, a E/S tal como é realizada no Patinho Feio. Neste capítulo, apenas os métodos para realizar a E/S serão examinados. As instruções correspondentes, e exemplos, serão vistos no capítulo 13.

Abaixo tem-se um esquema da interligação entre o computador e seus equipamentos de E/S.

Da figura acima tem-se a primeira regra (sem exceções): todos os dados que entram ou saem têm que passar pelo acumulador.

Ao Patinho Feio podem ser ligados 16 equipamentos de E/S (no máximo), cada qual ocupando um endereço de E/S, numerados de 0 a F (hexadecimal). Atualmente há 6 equipamentos, excluindo painel e duplex, ligados conforme a tabela abaixo:

Endereço de E/S	Equipamento	Tipo Entrada ou saída
0	Painel (RC)	E
5	Impressora (HP-2607A)	S
6	8-Bit duplex	E/S
7	8-Bit duplex	E/S
8	Perfuradora rápida de Fita de Papel	S
9	Leitora de Cartões	E
A	DECWRITER(Digital Equipment Corp.)	E/S
B	TTY (Teleprinter da TELETYPE Corp.)	E/S
E	Leitora de Fita de Papel	E

Os outros endereços de E/S ainda estão vagos.
A impressora é uma HEWLETT-PACKARD 2607A, com 132 colunas.
A DECWRITER é uma "máquina de escrever" com 72 colunas; o terminal da TELETYPE também, só que além disso, inclui leitora e perfuradora de fita de papel.
Os 2 endereços ocupados pela interface "8-Bit duplex" são para possibilitar a ligação entre o Patinho Feio e outros computadores.

Recomenda-se, para cada equipamento, consultar o manual fornecido pelo fabricante, para ter detalhes de operação do aparelho. Consultar também os apêndices deste manual, onde também se encontra o código ASCII, que é usado nos equipamentos.

Obs.: Os equipamentos de E/S efetivamente ligados ao Patinho Feio, vão sendo modificados à medida em que novos projetos são desenvolvidos pelo Laboratório de Sistemas Digitais. Para de ter uma relação atualizada desses equipamentos e mais detalhes de sua operação, recomenda-se consultar o manual "Resumo dos Equipamentos de E/S do Patinho Feio", publicado pelo Laboratório.

Estrutura dos Equipamentos de E/S (exceto painel)

Para fins de programação, todo equipamento de entrada e saída pode ser considerado como tendo:

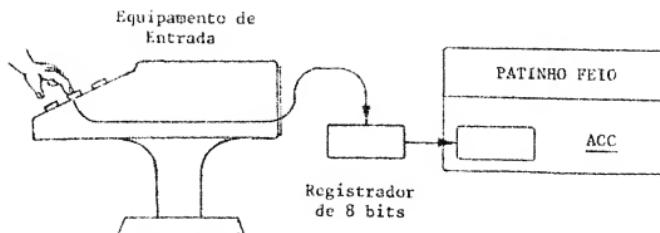
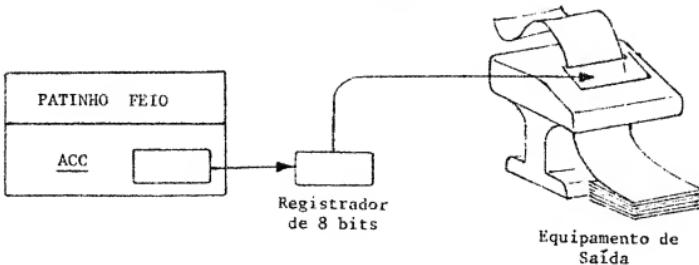
a) um registrador de 8 bits;

b) um flip-flop de controle;

c) um flip-flop de estado ("busy"(ocupado) ou "ready" (disponível));

d) um flip-flop de pedido de interrupção;

e) um flip-flop de "permite ou impede" pedidos de interrupção do dispositivo.

O painel só tem o registrador de 8 bits, chamado registrador de chaves, que fisicamente são as 8 chaves à direita (8 bits menos significativos) do RC completo, de 12 bits. Por esse registrador só é possível a entrada de dados. As instruções relativas ao painel estão discutidas no capítulo 9.

Obs.: Na realidade, todos os elementos citados acima estão em interfaces (circuitos que ligam os equipamentos de E/S ao Patinho Feio), mas isso não é relevante, do ponto de vista do programador. É mais fácil considerá-los diretamente como pertencentes ao equipamento em questão.

Em toda entrada de dados, o dado (8 bits) passa do meio exterior (isto é, cartão, fita perfurada, teclado, etc.), para o registrador de 8 bits do dispositivo correspondente. Completado esse processo, pode-se passá-lo desse registrador para o ACC.

Em toda saída de dados, o dado (8 bits) passa do ACC para o registrador de 8 bits do dispositivo correspondente, e daí para o meio exterior. Durante este último processo, não se pode mudar o conteúdo do registrador de 8 bits, pois senão haverá erro na saída.

Como é fácil ver, então, entram ou saem 8 bits por vez. A imagem no meio exterior desses 8 bits depende do dispositivo. Por exemplo, se se tratar da DECWRITER ou do terminal Teletype, que usam o código ASCII, ao número binário 0100 1011 corresponde a letra K do alfabeto. Portanto, se se mandar sair 0100 1011, o dispositivo escreverá a letra K no papel. Analogamente, se numa entrada de dados, bater-se na tecla K do teclado, o computador receberá o número 0100 1011. (No apêndice encontra-se o código ASCII completo). A saída ou entrada é, então, efetuada caracter por caracter.

A fita de papel perfurada que sai da perfuradora e é lida na leitora de fitas, tem 8 trilhas, correspondentes aos 8 bits da palavra, e cada coluna corresponde a um caracter. Se esse caracter for a letra K, na fita estará perfurado 0100 1011. Naturalmente, esta perfuração pode ser interpretada como sendo simplesmente o número /4B. Tudo depende de que espécie de dados foram colocados na fita, se números binários ou um "texto" codificado em ASCII.

Ver-se-á agora os dois possíveis métodos de E/S:

a) **Método "wait-for-flag":**

Consiste no seguinte: manda-se, por exemplo, entrar um dado. Ora, este dado deve passar do meio exterior para o registrador de 8 bits, e isto leva algum tempo. Não se prossegue o processamento, mas fica-se "perguntando" ao dispositivo: "— já acabou?" até que ele "responda": "sim". Neste instante pode-se passar o dado para o ACC e entrar mais um dado, ou prosseguir com o programa, se for o caso.(Por isso o método se chama "wait-for-flag", isto é, espera um aviso do dispositivo de que já acabou).

Na saída o método é análogo: passa-se o dado do ACC para o registrador de 8 bits do dispositivo conveniente, e manda-se passar o dado desse registrador para o meio exterior.Não se continua o programa, mas fica-se "perguntando" ao dispositivo: "já passou?", até que ele responda que "sim", quando, então, pode-se sair o próximo dado ou prosseguir o programa.

É necessário ficar fazendo essas perguntas, porque se a transferência do dado entre o meio externo e o registrador de 8 bits (ou vice-versa) não tiver acabado, não se pode ler o conteúdo do registrador de 8 bits (pois o que lá está não é o dado que queríamos!) na entrada; e, na saída, não se pode colocar ali outro dado (para não destruir o dado que está saindo!).

DIAGRAMA DE BLOCOS DO MÉTODO "WAIT-FOR-FLAG" PARA ENTRADA DE DADOS

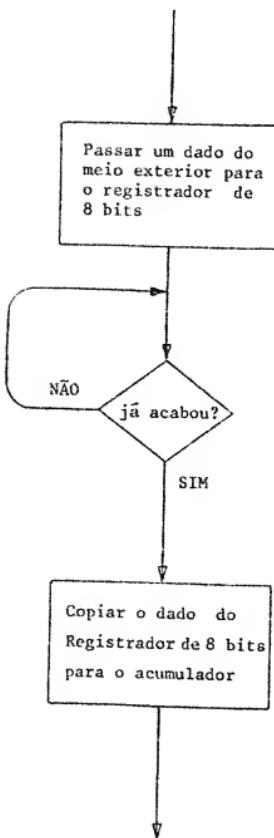
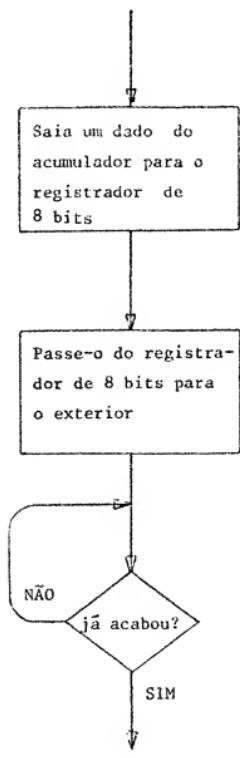
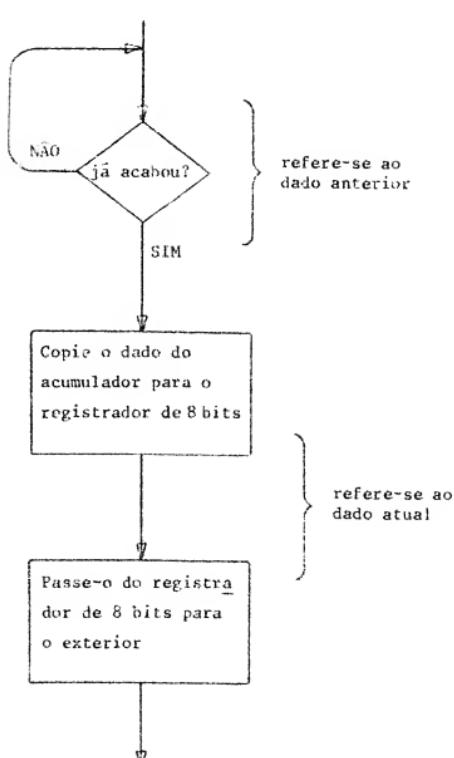



DIAGRAMA DE BLOCOS DO MÉTODO "WAIT-FOR-FLAG" PARA SAÍDA DE DADOS

O segundo método de saída é mais rápido que o primeiro, porque enquanto o dispositivo passa o dado para o meio exterior, pode-se continuar o processamento. Só depois de algum tempo, quando for necessária a saída do próximo dado, é que se testa se já acabou a saída do primeiro, e aí então, provavelmente, o dispositivo já terá acabado seu trabalho, ou pelo menos, o tempo de espera será menor.

b) Método de Interrupção:

Como se deve ter percebido da discussão do item anterior, o tempo que um equipamento de E/S leva para transferir um dado entre seu registrador de 8 bits e o meio exterior, é <u>muito maior</u> que o tempo de o Patinho Feio executar uma instrução qualquer. Logo, há um grande desperdício de tempo em ficar esperando o equipamento acabar sua tarefa, para prosseguir o processamento, como se faz no método "wait-for-flag" e, consequentemente, reduz-se a eficiência do programa.

Por outro lado, muitas vezes não é realmente necessário, na entrada principalmente, obter o dado do meio exterior imediatamente antes de usá-lo. Pode-se entrar o dado e deixá-lo em certa posição da memória, enquanto se faz outra coisa no programa, e depois vir buscá-lo nesta posição, quando ele se tornar necessário.

Para conseguir isso, existe a entrada e saída <u>por interrupção</u>. O conceito de interrupção é discutido no capítulo 11, e aqui só será vista sua aplicação à entrada e saída.

O método consiste no seguinte:

<u>Para a entrada</u>, ordena-se ao dispositivo que transfira o dado do meio exterior para o seu registrador de 8 bits, e que quando terminar o processo, peça uma interrupção ao Patinho Feio. Continua-se a processar o programa normalmente, até o ponto que for possível fazê-lo, sem ter o dado que queremos

entrar. Se até então, não tiver acontecido uma interrupção, pode-se executar uma instrução ESP, e deste modo, o Patinho Feio ficará esperando um pedido de interrupção.

O dispositivo, ao acabar a transferência do dado, pedirá uma interrupção. Supondo que esta esteja permitida, haverá um desvio para a posição 4 da memória, onde deve haver uma rotina para tratamento da interrupção, que geralmente faz o seguinte:

- salva o estado da máquina, se necessário;
- determina qual dispositivo pediu interrupção (principalmente se se estiver trabalhando com esse método em mais de um equipamento de E/S, porque sabe-se apenas que houve interrupção, mas não de onde proveio o pedido — pode ter sido até do botão do painel!);
- torna o dado disponível para o programa (isto é, armazena-o numa posição conveniente ou no ACC);
- pode, conforme o caso, atualizar contadores e outras variáveis do programa;
- restaura estado da máquina, se necessário;
- volta ao processamento normal.

Quando dois dispositivos pedirem interrupção simultaneamente, é necessário escolher um deles para ser tratado em primeiro lugar. Geralmente, escolhe-se o dispositivo mais rápido (no Patinho Feio, é a leitora de fita e, futuramente, será o disco).

Para a saída, o método é inteiramente análogo, isto é, passa-se o dado do ACC para o registrador de 8 bit do dispositivo e manda-se executar a transferência para o meio exterior. Prossegue-se normalmente o programa, até haver uma interrupção do equipamento, quando este acabar a transferência.

Compreende-se facilmente que, com este método, a E/S de dados pode tornar-se muito mais rápida que com o método de "wait-for-flag". Contudo, ele é menos usado pois exige mais tempo de programação, pois é necessário fazer uma rotina de tratamento de interrupção que pode, além disto, tornar-se bastante extensa (dependendo do programa). Por isso, geralmente, impede-se que seja pedida interrupção, e usa-se o método de "wait-for-flag". Como fazer isso, será visto em seguida.

Funções dos flip-flops dos equipamentos de E/S

Como se recorda, do ponto de vista do usuário, todo equipamento de E/S tem, além do registrador de 8 bits, 4 flip-flops, cuja utilidade é descrita abaixo:

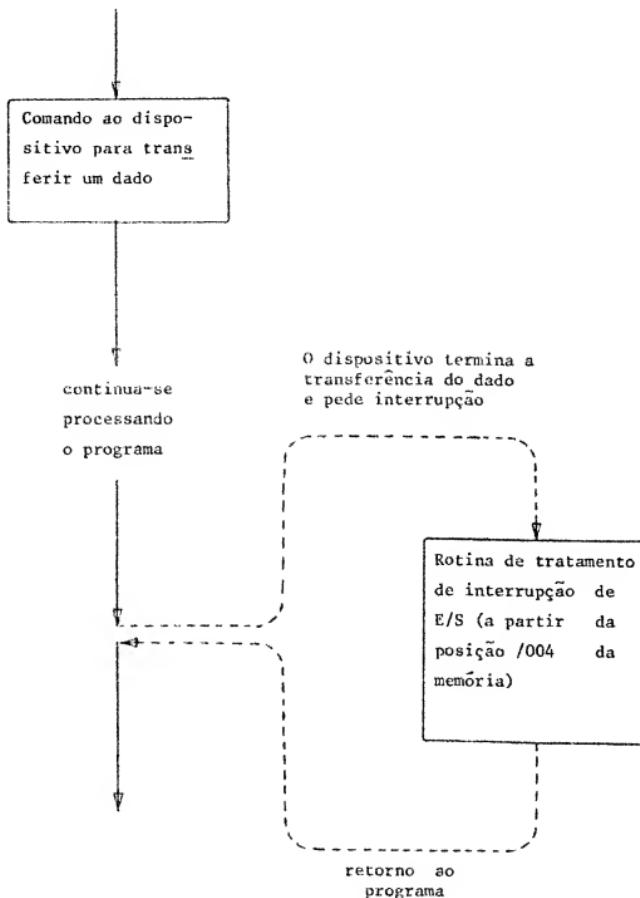
1) flip-flop de CONTROLE - como o próprio nome diz, controla o funcionamento do dispositivo. O controle ligado é a ordem para o dispositivo transferir o dado entre o registrador de 8 bits e o meio exterior ou vice-versa, conforme o caso. Quando a transferência termina, o controle é automaticamente desligado;

2) flip-flop de ESTADO - indica o estado do dispositivo: se este flip-flop estiver desligado, o dispositivo está ocupado ("busy") com uma transferência de dados; se ele estiver ligado, a transferência acabou e o dispositivo está disponível ("ready"). É este flip-flop que se usa no método "wait-for-flag" para saber se já acabou a transferência do dado . Portanto, antes de ligar o flip-flop de controle, é necessário desligar o flip-flop de estado para indicar que o equipamento está ocupado ("busy"). A seguir fica-se testando esse flip-flop.
   O equipamento, ao acabar a transferência do dado, liga-o automaticamente e, então, fica-se sabendo que ele está novamente disponível ("ready" e que a transferência acabou;

3) flip-flop de PEDIDO DE INTERRUPÇÃO - quando está ligado, manda um pedido de interrupção ao Patinho Feio (ver diagrama). Este flip-flop é ligado pelo flip-flop de estado, isto é, quando aquele estiver ligado vai ligar este flip-flop também;

4) flip-flop que PERMITE/IMPEDE (conforme esteja ligado ou não) que o dispositivo faça um pedido de interrupção ao Patinho Feio. Quando se utiliza o método de E/S "wait-for-flag", faz-se com que este flip-flop impeça interrupções do dispositivo de E/S.

Quando se faz a entrada e saída com interrupção, o que ocorre é que quando o equipamento termina a transferência do dado, automaticamente desliga o flip-flop de controle e liga o de estado. Este, por sua vez, liga o PEDIDO DE INTERRUPÇÃO, se o flip-flop PERMITE/IMPEDE permitir, que se supõe seja aceito. Então o Patinho Feio passa a processar a interrupção. Ora, não se pode terminar a interrupção simplesmente com a instrução PUL, pois o PUL limpa o estado de interrupção do sistema, mas não desliga o pedido de interrupção que vem do equipamento de E/S. Isso ocasionaria nova interrupção, não desejada, logo a seguir. Portanto, é necessário desligar o flip-flop de pedido de interrupção do dispositivo que já foi tratado e, antes disso, também o seu flip-flop de estado, pois se este permanecer ligado, irá religar o pedido de interrupção logo a seguir.

Ver-se-á agora como fazer quando for usado o método da interrupção com vários equipamentos, e mais de um pedir interrupção.

Como se recorda, o Patinho Feio tem apenas um nível de interrupção. Isto significa que, se se estiver tratando uma interrupção de um equipamento de E/S e um outro pedir interrupção, o pedido não será aceito antes do fim da primeira interrupção.

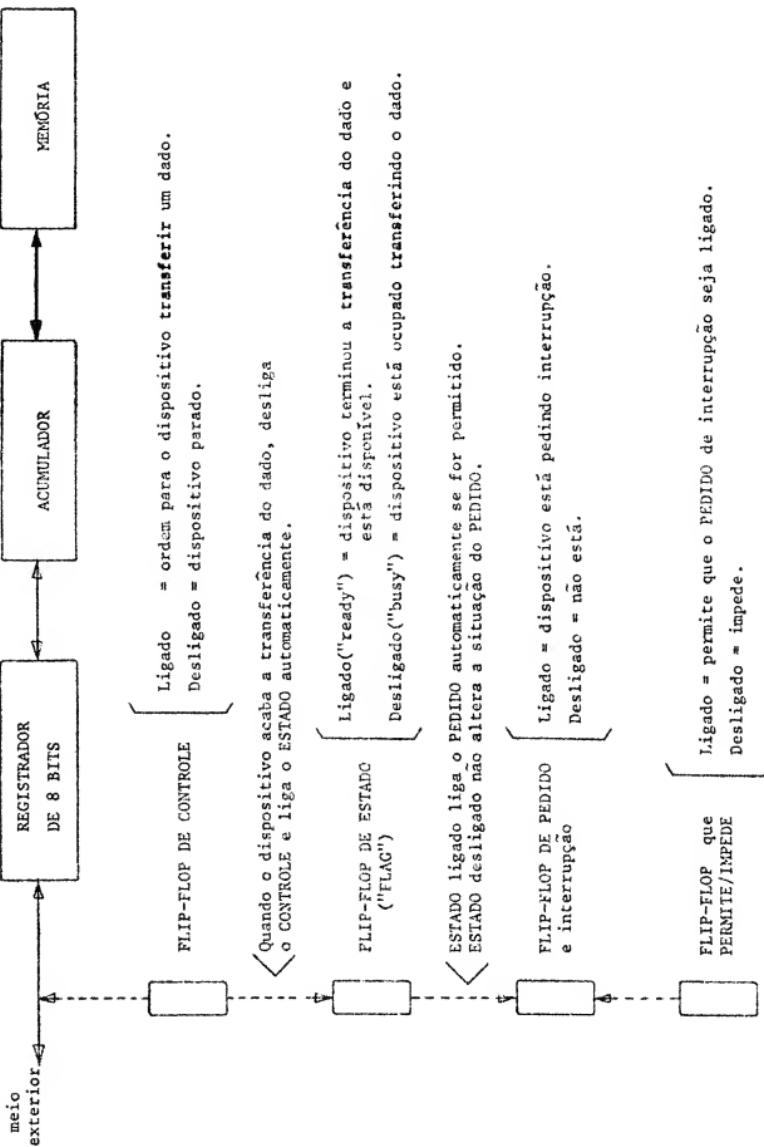
Isto sugere dois métodos para tratar estes casos:

a) Quando houver uma interrupção, testar todos os dispositivos para ver qual ou quais a pediram. Tratar todos os pedidos existentes, um a um, e após cada tratamento, desligar os flip-flops de estado e de pedido de interrupção do dispositivo tratado. Terminados todos os dispositivos, encerrar a interrupção.

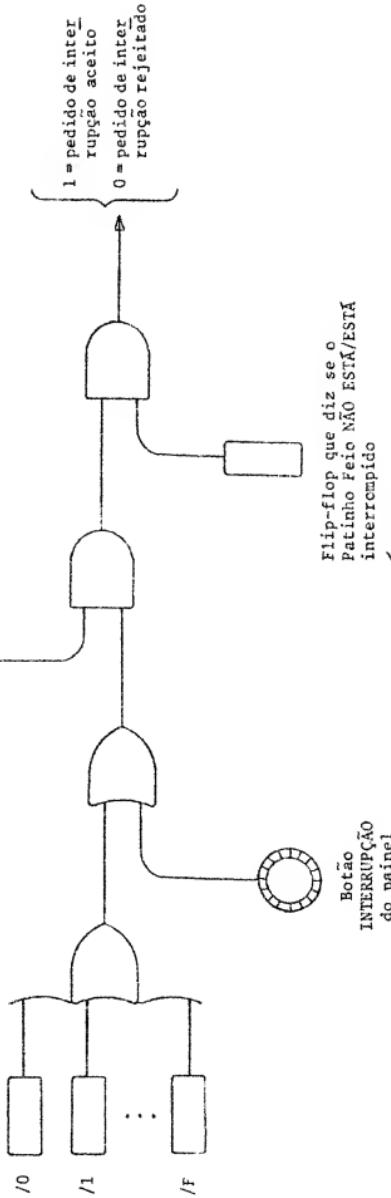
b) Testar os dispositivos segundo sua prioridade. Achando um que pediu interrupção, tratá-lo. Em seguida, desligar os flip-flops de estado e de pedido de interrupção desse dispositivo e encerrar a interrupção com o PUL. Se houver mais algum equipamento pedindo interrupção, haverá nova interrupção logo em seguida e outro dispositivo será tratado.

Em geral, o método a é mais rápido, pois não é necessário cada vez ficar salvando e restaurando o estado da máquina no momento da interrupção, como é feito no método b. Contudo, o programa para b é mais fácil de se fazer, e é menor.

Diagrama de Blocos do método de entrada e saída por interrupção:

Observação: Não é possível determinar "a priori" em que ponto do programa acontecerá o pedido de interrupção.

Para finalizar este capítulo, tem-se, a seguir, um diagrama geral dos diversos registradores e flip-flops usados para comandar a E/S por interrupção e por "wait-for-flag". Cada quadrado é um elemento ("flip-flop") de dois estados (0 e 1) cujos nomes dependem do particular elemento. Flechas contínuas indicam fluxo de informação. Flechas pontilhadas indicam influência de um elemento sobre outro.

A seguir há um diagrama lógico que mostra quando e como o Patinho Feio aceita um pedido de interrupção. O diagrama usa blocos lógicos AND e OR, explicados no capítulo 2.

Evidentemente, só um dispositivo de E/S foi representado no esquema, cuja parte referente ao dispositivo deve ser repetida para cada equipamento a ser usado.

Através do diagrama lógico é fácil ver que uma interrupção de um equipamento de E/S só é possível quando:

- o Patinho Feio ainda não estiver em interrupção;
- a interrupção não estiver inibida (pela instrução INIB);
- a instrução não estiver impedida pelo flip-flop impede/permite;
- e houver pedido de interrupção do dispositivo em questão.

meio exterior
REGISTRADOR DE 8 BITS
ACUMULADOR
MEMÓRIA
FLIP-FLOP DE CONTROLE
Ligado = ordem para o dispositivo **transferir** um dado.
Desligado = dispositivo parado.
Quando o dispositivo acaba a transferência do dado, desliga o CONTROLE e liga o ESTADO automaticamente.
FLIP-FLOP DE ESTADO ("FLAG")
Ligado("ready") = dispositivo terminou a **transferência** do dado e está disponível.
Desligado("busy") = dispositivo está ocupado **transferindo** o dado.
ESTADO ligado liga o PEDIDO automaticamente se for permitido.
ESTADO desligado não altera a situação do PEDIDO.
FLIP-FLOP DE PEDIDO e interrupção
Ligado = dispositivo está pedindo interrupção.
Desligado = não está.
FLIP-FLOP que PERMITE/IMPEDE
Ligado = permite que o PEDIDO de interrupção seja ligado.
Desligado = impede.

Flip-flops de PEDIDO de interrupção dos dispositivos
/0
/1
/F
Botão INTERRUPÇÃO do painel
Flip-flop que PERMITE/INIBE interrupção do Patinho Feio
Ligado = permite interrupção.
Desligado = inibe interrupção.
Flip-flop que diz se o Patinho Feio NÃO ESTÁ/ESTÁ interrompido
Ligado = não está interrompido.
Desligado = está interrompido.
1 = pedido de interrupção aceito
0 = pedido de interrupção rejeitado

As instruções que mexem nos flip-flops de CONTROLE, ESTADO, PEDIDO e PERMITE/IMPEDE, são instruções de E/S e serão vistas no próximo capítulo.

O flip-flop PERMITE/INIBE é desligado pela instrução INIB e ligado pela instrução PERM (capítulo 11).

O flip-flop NÃO ESTÁ/ESTÁ é desligado pelo Patinho Feio, ao aceitar uma interrupção, e religado, também pelo Patinho Feio, ao término da interrupção (instrução PUL).

Ao ser apertado o botão "preparação" no modo endereçamento (ver capítulo 14), os flip-flops tomam o seguinte conteúdo:

CONTROLE	desligado
ESTADO	desligado
PEDIDO	desligado
PERMITE/IMPEDE	impede
PERMITE/INIBE	permite
NÃO ESTÁ/ESTÁ	não está

Com isso, foram vistos os métodos para realizar E/S no Patinho Feio. As instruções para executar esses métodos, bem como exemplos, serão vistos no capítulo seguinte (capítulo 13).

13 - INSTRUÇÕES DE E/S

Todas as instruções de E/S são longas, isto é, ocupam duas palavras da memória. A primeira palavra começa com o hexadecimal /C. Existem quatro tipos de instruções de E/S: FNC, SAL, SAI, ENTR.

O operando destas instruções é uma constante que, quando convertida para o formato hexadecimal, é da forma /nc , onde n indica o canal de E/S e c o comando (isto é, a ação a executar), que é o último dígito hexadecimal da instrução. Nenhuma instrução de E/S altera os conteúdos de V e T.

Instruções FNC:

Código de Máquina	Instrução	Descrição
Cn 10	FNC /n0	Desliga flip-flop PERMITE/IMPEDE para o dispositivo n (isto é, impede interrupção do dispositivo n).
Cn 11	FNC /n1	Desliga flip-flop de ESTADO do dispositivo n (ESTADO = "busy").
Cn 12	FNC /n2	Liga flip-flop de ESTADO do dispositivo n (ESTADO = "ready").
Cn 14	FNC /n4	Desliga flip-flop de PEDIDO de interrupção do dispositivo n.
Cn 15	FNC /n5	Liga flip-flop PERMITE/IMPEDE para o dispositivo n (isto é, permite interrupção do dispositivo n).
Cn 16	FNC /n6	Liga flip-flop de CONTROLE e desliga flip-flop de ESTADO (ESTADO = "busy") do dispositivo n .

Código de Máquina	Instrução	Descrição
Cn 17	FNC /n7	Desliga flip-flop de CONTROLE do dispositivo n.
Cn 18	FNC /n8	Só funciona na leitora de fita, canal /E. Ignora todos os "feed-frames" ("bytes" nulos) da fita, até a próxima perfuração (1º "byte" não nulo).

Instruções SAL: salta duas palavras se:

Código de Máquina	Instrução	Descrição
Cn 21	SAL /n1	o flip-flop de ESTADO do dispositivo n estiver ligado.
Cn 22	SAL /n2	o dispositivo n estiver O.K. (ver observação abaixo).
Cn 24	SAL /n4	o flip-flop de PEDIDO de interrupção do dispositivo n estiver desligado.

Obs.: A instrução SAL /n2 só funciona para os dispositivos: 5, 8, E, de E/S.

Dispositivo 5 (impressora) - salta se houver papel na impressora e esta estiver pronta para imprimir.

Dispositivo 8 (perfuradora de fita) - salta se houver fita na perfuradora e esta estiver pronta para perfurar ("on-line").

Dispositivo E (leitora de fita) - salta se estiver com a fita a ser lida instalada e a leitora estiver "on-line".

Nos outros dispositivos, não salta.

Instrução ENTR:

Código de máquina: /Cn 40

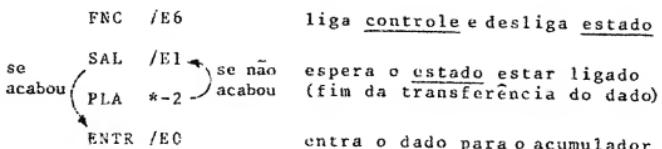
Instrução: ENTR /n0

Descrição: Entrada do dado do registrador de 8 bits do dispositivo n para o acumulador.

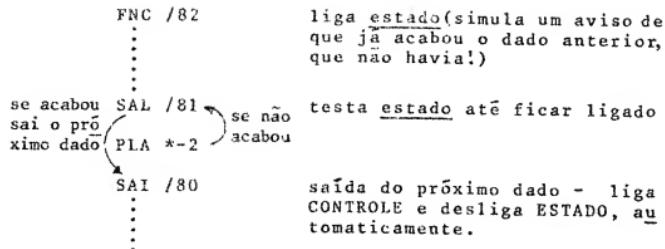
Instrução SAI:

Código de máquina: /Cn 80

Instrução: SAI /n0

Descrição: Saída do dado do acumulador para o registrador de 8 bits do dispositivo n. A seguir,liga flip-flop de CONTROLE e desliga flip-flop de ESTADO (ESTADO = "busy") do dispositivo n, automaticamente(o que causa a saída do dado para o meio exterior).

Repete-se aqui a lista dos dispositivos em uso no Patinho Feio:

Endereço de E/S	Equipamento	Tipo
5	Impressora	Saída
8	Perfuradora de Fita de Papel	Saída
9	Leitora de Cartões	Entrada
A	DECWRITER	Entrada e Saída
B	Teleprinter(TTY)	Entrada e Saída
E	Leitora de Fita de Papel	Entrada

<u>Exemplos de E/S</u>: (ver também capítulo 12)

a) "wait-for-flag":

a.1) entrada de um caracter na leitora de fita(canal E):

```
                 FNC   /E6              liga controle e desliga estado
se               SAL   /E1 <- se não    espera o estado estar ligado
acabou           PLA   *-2 -  acabou    (fim da transferência do dado)
                 ENTR  /E0              entra o dado para o acumulador
```

a.2) saída de um dado pela perfuradora de fita(canal 8)pelo segundo método:

Início do Programa

b) interrupção:

b.1) aceita um caracter pela DECWRITER (dispositivo /A)e armazena na matriz A. Além disso, se for um "P", soma um ao contador CONT. Se for um "E", imprime na TTY (dispositivo /B), usando "wait-for-flag" - 1º método (capítulo 12).

Programa Principal

```
        ORG     ....
          :
CONT    DEFC    0     valor inicial de CONT.
          :
        FNC     /A1   desliga flip-flop de estado do dispo-
                      sitivo /A,pois ele ligado vai  ligar
                      o PEDIDO.
        FNC     /A4   desliga flip-flop de pedido de inter-
                      rupção.
        FNC     /A5   permite interrupção do dispositivo /A

        FNC     /A6   liga flip-flop de controle da DECWRI-
                      TER, para entrada do 1º dado.
          :
          :     resto do programa principal
          :
```

Rotina de Tratamento de Interrupção:

```
        ORG     /4

        ARM     SAVE    salva estado da máquina no instante da
                        interrupção
        ENTR    /A0     entra o dado para o acumulador

        ARMX    A       armazena na matriz A (indexação)

        SOMI    -@P
                        testa se é P
        PLAZ    EPE

        CARX    A       não é P

        SOMI    -@E
                        testa se é E
        PLAZ    EE

AQUI    TRI     não é E,      Se é E    EE   CARI   @E
        INC     soma um                      SAI    /B0    imprime E
                  ao
        TRI     indexador                    SAL    /B1    na TTY
                                             PLA    *-2
                                             PLA    AQUI
```

```
              CARI  0
                              restaura estado da máquina
        SAVE  EQU   *-1
              FNC   /A1       desliga estado
              FNC   /A4       desliga pedido
              FNC   /A6       liga controle,para entrada de novo dado
              PUL             fim da interrupção

Se é P  EPE   CAR   CONT
              INC             soma um a CONT
              ARM   CONT
              PLA   AQUI
```

14 - PROGRAMAS ABSOLUTOS E RELOCÁVEIS

Neste capítulo discutir-se-á um conceito que aumenta enormemente os recursos de programação disponíveis em um computador: o de programas relocáveis. É um conceito puramente da área de "software" (programação), já que não envolve nenhuma nova instrução executável pela máquina.

Por outro lado, surgem novas pseudo-instruções (para o montador), que serão discutidas no próximo capítulo.

Examine-se, de início, um pequeno programa, que aceita caracteres da DECWRITER (dispositivo /A) e perfura-os numa fita de papel (dispositivo /8), até encontrar uma vírgula, que indica o fim dos dados e não é perfurada.

```
@TLB
      ORG   /472    indica o endereço onde começa-se a arma
                    zenar dados ou o programa.
INI   FNC   /A6
      SAL   /A1
      PLA   *-2     lê caracter na DEC("Wait-for-flag")
      ENTR  /A0
      ARM   DADO    e guarda em DADO
      SOMI  -@,
                    testa se é uma vírgula
      PLAZ  CAB
      CAR   DADO    não é, portanto
      SAI   /80
                    perfura o caracter("Wait-for-flag"- mé-
      SAL   /81     todo 1) (capítulo 12)
      PLA   *-2
      PLA   INI     e vai ler próximo caracter.
```

CAB	PARE		caracter lido é, (fim do programa)
DADO	DEFC	0	posição para armazenar caracter lido.
	FIM	INI	endereço de início de execução do programa.

Ao olhar o programa acima, escrito para o montador absoluto, pode-se notar um fato interessante. A origem /472 não tem a menor influência no resto do programa. O que ocorreria se fosse, por exemplo, ORG /AB6 em vez de ORG /472 ?

Ocorreria apenas uma mudança dos locais físicos da memória ocupados pelo programa, mas o seu funcionamento continuaria exatamente igual, e correto.

Já no caso, por exemplo, de uma interrupção, a rotina de tratamento correspondente deve começar,obrigatoriamente, na posição /4 da memória. Não daria certo o programa se fosse colocado a partir de uma outra posição qualquer.

Retornando ao programa-exemplo: é uma boa idéia fazer com que a origem torne-se, por enquanto, indefinida, pois ela não é essencial ao programa. Para não se confundir nas instruções do programa, atribui-se (arbitrariamente) à primeira palavra do programa, o número <u>zero</u> e numera-se as outras palavras <u>relativamente</u> ao início do programa. Fica-se, então, com a seguinte numeração (lembrando que as instruções longas ocupam duas palavras):

Número (hexadecimal)	Instrução		
000	INI	FNC	/A6
002		SAL	/A1
004		PLA	*-2
006		ENTR	/A0
008		ARM	DADO
00A		SOMI	-@,
00C		PLAZ	CAB
00E		CAR	DADO
010		SAI	/80
012		SAL	/81
014		PLA	*-2
016		PLA	INI
018	CAB	PARE	
01A	DADO	uma palavra para DADO	

O número chama-se <u>endereço relativo</u> (ao início do programa) da instrução.

Tem-se a seguinte tabela de símbolos e seus respectivos endereços <u>relativos</u>:

INI	000	
CAB	018	
DAO	01A	(DADO, truncado para 3 letras)

Além disso, na instrução de endereço relativo /004, que é PLA *-2, o operando *-2 refere-se então ao endereço relativo /002.

Analogamente, na instrução /014, o operando *-2 refere-se ao endereço relativo /012.

O programa está agora com uma origem "flutuante", isto é, pode-se atribuir qualquer valor α à origem.

Isto feito, a instrução FNC /A6 terá endereço absoluto (ou seja, endereço da posição física da instrução na memória) igual a α ; SAL /A1 terá endereço absoluto α + 2; PLA *-2 terá endereço absoluto α + 4, e assim por diante.

Também, dos endereços relativos dos símbolos, decorrem imediatamente os endereços absolutos correspondentes:

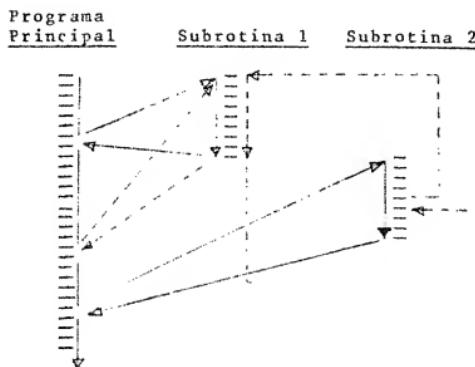
INI	α + /000
CAB	α + /018
DAO	α + /01A

Está agora terminada a transformação do pequeno programa-exemplo, de absoluto em relocável, isto é, que pode ser mudado de lugar dentro da memória.

Vê-se, então, que o conceito de programa relocável é bem simples. Quase sem mudanças no programa, ele foi transformado em relocável. A função do montador relocável é agora pegar este programa-fonte e transformá-lo em um código de máquina, com os endereços numerados relativamente ao início do programa, como aqui foi feito. A saída gerada pelo montador relocável ainda não é executável pelo computador, por causa do fato de o endereçamento ser relativo. É necessário então mais um passo, onde um programa(chamado carregador relocável ou relocador-ligador) pega como dados a saída do montador relocável e um endereço de origem,que pode ser dado pelo programador,e calcula todos os endereços absolutos, gerando então, por sua vez, o código de máquina já em formato executável (ver também o capítulo 16).

Se a única aplicação do conceito de relocação fosse esta que acabou de ser mostrada no programa-exemplo, ele não seria tão importante. Mas, a verdadeira importância da relocação será vista a seguir, quando forem vistos novos conceitos e aplicações da relocação.

Tipos de Programas:

Considere-se agora, não um pequeno programa como no exemplo dado, mas um programa enorme, talvez tão grande que não caiba na memória. É então muito conveniente, se não mesmo imperativo, dividir o programa em várias rotinas, cada uma fazendo uma ou várias funções do programa original, e executar uma de cada vez. Evidentemente, é necessário, ao passar de uma rotina para outra, levar em conta o trabalho já feito e não destruir o que já foi calculado.

Quando se faz a subdivisão indicada acima, obtém-se rotinas, que têm vários nomes, de acordo com a função a que destinam ou o modo como são encaradas: programa principal, subrotina e segmento.

Programa Principal é, como o próprio nome diz, a parte principal do que se quer fazer. É a parte do programa que controla a sequência de operações a serem executadas. Por exemplo, o programinha feito no início do capítulo é um programa principal, que, no caso, é todo o programa.

Evidentemente, não pode haver mais de um programa principal num conjunto de rotinas a serem executadas, pois só uma rotina pode controlar todas as outras.

Subrotina é uma rotina subordinada a um outro trecho de programa. Em geral, a subrotina é bastante menor que um programa principal ou segmento, e é chamada várias vezes durante a execução, de vários locais distintos do programa. Após a subrotina fazer seu trabalho, o controle retorna ao ponto de onde proveio a chamada. Além disso, nada impede que uma subrotina chame outra (subrotinas encaixadas).

Exemplo:

Programa Principal chama:

Subrotina 1   2 vezes

Subrotina 2   1 vez

e subrotina 2 chama:

subrotina 1   uma vez

Naturalmente, podem existir também, subrotinas dentro de uma divisão do programa, executáveis por uma instrução PUG (cap.5). Aqui está-se tratando, contudo, de subrotinas que são montadas separadamente, e que são independentes das outras unidades do programa, podendo, inclusive, serem aproveitadas em vários programas.

Se se dispuser de uma biblioteca de subrotinas desse tipo, que executem várias tarefas frequentemente necessárias (como operações aritméticas, operações de E/S, preenchimento de áreas da memória com zeros ou outro valor, etc), o trabalho de fazer um programa pode ser bastante reduzido.

Além disso, a divisão de um programa em várias subrotinas, apresenta a enorme vantagem de cada uma delas poder ser corrigida e melhorada, independentemente das outras,o que acelera e facilita muito este trabalho. O mesmo se dá com os segmentos, que serão vistos logo mais.

Um programa principal bem feito e bem estruturado poderá então consistir de quase nada mais que uma sucessiva chamada de subrotinas, cada uma para executar uma diferente tarefa.

Um Segmento é uma rotina que apresenta características tanto de programa principal como de subrotina. É um trecho de um grande programa, que é executado,em geral, uma ou duas vezes, e que, de costume, não é chamado de outro ponto e nem a ele retorna. Sua execução se dá ao terminar o segmento anterior-e quando termina, passa-se ao próximo segmento. Em geral, usa-se em grandes programas, quando o programa principal e respectivas subrotinas não cabem simultaneamente na memória. Naturalmente, um segmento pode chamar subrotinas, se necessário, e ao seu término, o controle volta ao ponto de chamada no segmento.

Note-se que não é necessário haver um programa principal: pode-se substituí-lo por vários segmentos.Começa-se executando o 1º segmento; ao terminá-lo passa-se a executar o 2º segmento, e assim por diante, até acabar a execução do último segmento do programa.

Por outro lado, pode-se ter um programa cujo processamento pode seguir por vários caminhos, dependendo de condições que só ocorrerão no instante da execução. Se, para cada ocorrência, o processamento for bastante longo, pode ser conveniente fazer um segmento para cada possível ocorrência e um programa principal que identifique as ocorrências e controle então a execução do segmento correspondente.

Diagrama esquemático:

programa (tudo o que se quer fazer)

memória do Patinho Feio

Dividindo em várias partes:

Programa Principal

segmento 1 | subrotina 1
segmento 2 | subrotina 2
. | .
. | .
. | .
segmento m | subrotina n

Deste modo, com a técnica de segmentação, conseguir-se-á executar tudo o que originalmente era desejado, o que de outro modo não seria possível. Somente uma parte do "programão" estará na memória em um dado instante — por exemplo, o programa principal, um segmento e algumas subrotinas. O resto estará armazenado em algum meio externo; por exemplo, uma fita perfurada de papel ou, no futuro, em um disco magnético.

Tipos de variáveis e endereços

Pode-se notar que, devido à maior complexidade dos métodos acima em relação àqueles do montador absoluto, é necessário um cuidado bem maior no tratamento de variáveis, para não haver, acidentalmente, perda de informações. Por isso existe já uma divisão das variáveis e endereços, de acordo com o seu tipo, conforme será visto a seguir. Essa divisão, além de sistematizar o tratamento das variáveis, facilita a compreensão de várias características da relocação e simplifica a elaboração de programas.

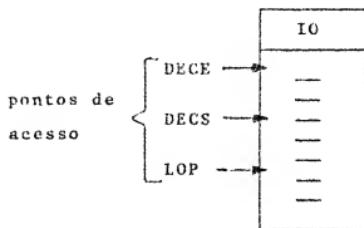
As variáveis, rótulos (identificadores), símbolos e endereços que podem existir num programa relocável, são de um dos seguintes tipos: absolutos, relocáveis, pontos de acesso, externos e comuns (do inglês "common").

Um endereço absoluto é, como o próprio nome diz, o endereço de uma posição física (real) da memória. Ou seja, não é uma posição numerada relativamente ao início do programa, mas sim ao início da memória. Mesmo que o programa seja posto em vários locais diferentes na memória, um endereço absoluto referenciará sempre a mesma posição física da memória.

O melhor exemplo para o que foi explicado acima é o indexador. Se se quiser conhecer seu conteúdo, tem-se que endereçar a posição /000 da memória, qualquer que seja o local onde

for armazenado o programa relocável. Se se tiver, por exemplo, a instrução CAR IDX, onde IDX é o símbolo que queremos associar ao indexador, tem-se que conseguir que IDX se refira à posição absoluta /000 da memória, e não, evidentemente, ao endereço zero do programa relocável (1a. instrução), que ocupará uma posição ainda não determinada na memória.

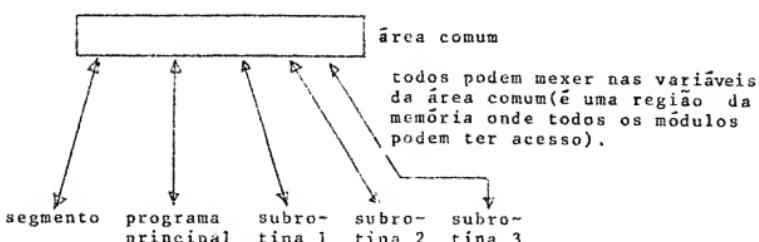
Em contraposição, um endereço relocável é aquele relativo ao início do programa, e cuja posição final absoluta na memória dependerá, mais tarde, da origem associada ao programa. As variáveis relocáveis (ou seja, associadas a endereços relocáveis) são variáveis locais, no sentido de que só "valem" dentro da unidade do programa (subrotina, segmento ou programa principal) em que se estiver trabalhando. Uma vez que se tenha começado outra subrotina ou segmento, as variáveis relocáveis de outras rotinas do programa não são mais a ela acessíveis.

Não há dificuldade em se pensar em exemplos de variáveis relocáveis, pois elas,em geral, constituem a maioria das variáveis encontradas numa unidade de um programa. No exemplo apresentado no início do capítulo, todos os símbolos são relocáveis.

Os símbolos globais externos permitem a uma rotina de um programa ter acesso, por nome, a variáveis definidas em outras rotinas, e também permitem a chamada de subrotinas não incluídas na unidade corrente do programa. Tratam-se de símbolos que, embora referenciados em instruções da unidade corrente do programa, estão definidos em outra unidade. É então necessário dizer ao montador relocável que se tratam de símbolos externos, caso contrário, ele vai procurar a definição do símbolo na unidade atual do programa e, não a encontrando, vai dar a mensagem de erro: "símbolo indefirido".

Os símbolos externos também são usados na chamada de subrotinas e execução de segmentos, da seguinte forma: na instrução PUC <nome> ou PLA <nome>, <nome> é símbolo externo, indicando desvio para uma posição que não se encontra na atual rotina.

Os *pontos de acesso* ("entry points") são justamente os locais de uma rotina para onde se pode efetuar um desvio a partir de uma outra rotina. Eles marcam *posições dentro* de uma rotina, e por isso mesmo referem-se a endereços *relocáveis*, que ficam identificados pelo nome do ponto de acesso.

Por exemplo, suponha-se que uma subrotina chamada IO realiza entrada e saída de dados. Ela pode ter, por exemplo, três pontos de entrada, um chamado DECE para entrada de dados pela DECwriter, outro chamado DECS para saída de dados pela DECwriter, e, por fim, um chamado LOP para entrada pela leitora de fita.

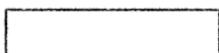
subrotina

Quando, de uma outra rotina, se quiser efetuar uma entrada de dados pela leitora de fita, far-se-á um PUC LOP, e desta forma começar-se-á a subrotina IO diretamente pelo ponto de acesso LOP, não executando as instruções que vem antes e que não nos interessam. Evidentemente, LOP deve ser declarado *externo* na rotina que chama, pois só está definido em outra subrotina.

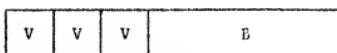
Por fim, uma variável "<u>common</u>" é uma variável comum a várias rotinas, isto é, é uma posição da memória que pode ser referenciada e ter seu conteúdo alterado por várias unidades de um programa. Em contraposição tem-se as variáveis relocáveis , que só podem ser acessadas de dentro de sua própria unidade.

Em geral, usam-se em um programa, várias variáveis comuns ("common"), que são então agrupadas em uma <u>área</u> de variáveis comuns. Note-se que as variáveis da área comum não tem obrigatoriamente o mesmo nome nas várias rotinas que compõem o programa, embora se refiram à mesma posição de memória.

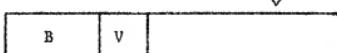
As variáveis (inclusive blocos, ou seja, matrizes) , que se quer colocar na área comum, devem ser declaradas com os respectivos nomes em cada rotina do programa. Não é obrigatório que o <u>comprimento</u> da área comum (em número de palavras de memória ocupadas) seja igual em todas as rotinas, mas é <u>necessário</u> que o maior comprimento de todas as áreas comuns declaradas seja o do programa principal. Não é permitido em uma subrotina ou segmento declarar uma área comum de comprimento maior que a do programa principal. A razão para isto será vista mais tarde (vide <u>divisão da memória</u>).

Os esquemas abaixo facilitarão a compreensão dos conceitos de variável comum e divisão da área comum entre as várias variáveis e seus nomes nas rotinas.

No esquema abaixo, V = variável, B = Bloco (matriz), e na vertical está o nome dado à variável ou ao bloco na respectiva rotina.

Áreas relocáveis (regiões distintas da memória, cada uma contendo o código-objeto e as variáveis locais a cada rotina).

Programa Principal

Subrotina um

Subrotina dois

Área Comum (a mesma região de memória para todas as rotinas, mas dividida diferentemente em cada uma delas, contém as variáveis declaradas como sendo comuns).

Como se pode notar, a área comum foi dividida nas várias rotinas de modos diferentes, embora isto não seja obrigatório. Assim, por exemplo, as variáveis PTA, KTC, ABA e parte do bloco UTX do programa principal, ocupam as mesmas posições de memória que o bloco FUJ na subrotina um. Uma posição de memória colocada na área comum no programa principal sob o nome de ABA é, na subrotina 2, conhecida sob o nome de ISS, pois é como ISS que ela foi posta na área comum, na subrotina 2.

Além disso, existem partes da área comum completa, que não podem ser referenciadas diretamente nas subrotinas 1 e 2, simplesmente por não terem sido declaradas. Poderão, talvez, ser referenciadas com um deslocamento.

Para aplicação dos conceitos vistos é dado o exemplo abaixo, com o esquema geral de uma subrotina que calcula o seno de um ângulo x em radianos.

Suponha-se que estejam disponíveis duas rotinas:

- FLUTA, que soma ou subtrai números em ponto flutuante, com os respectivos pontos de acesso ADD e SUBT. Dados em A e B, resposta em C, e não altera os valores de A e B.

- FLUTB, que multiplica ou divide números em ponto flutuante, com os respectivos pontos de acesso MPY e DIV. Dados em A e B, resposta em C, e não altera os valores de A e B.

Estas subrotinas são necessárias porque o Patinho Feio só maneja números inteiros de -128 a +127. Portanto, para se usar números reais ("ponto flutuante"), é necessário escrever programas adequados.

Será usada a seguinte aproximação:

$$\text{sen } x \simeq x - \frac{x^3}{3!} + \frac{x^5}{5!}$$

Deve-se:

- obter o valor de x e colocá-lo na posição X ;
- multiplicar x por x , obtendo x^2 ;
- multiplicar x^2 por x, obtendo x^3 ;
- dividir x^3 por 6 (=3!) ;
- subtrair de x ;
- multiplicar x^2 por $\frac{x^3}{6}$;
- dividir por 20 ;
- somar ao resultado anterior ;
- voltar da subrotina.

Serão colocadas na área comum as variáveis A, B, C, X, SIN, pois são mexidas por diversas subrotinas.

Sequência de operações a efetuar (lembrar-se que(...) denota o conteúdo de uma posição):

1º) colocar x na posição X ;

2º) colocar x em A e em B ;

3º) chamar subrotina FLUTB, no ponto de entrada MPY (PUG MPY) ; } calcula x^2

4º) passar C = x^2 para A ;

5º) PUG MPY (calcula x^3) ;

6º) passar A = x^2 para a posição XDO, para uso posterior;

7º) passar C = x^3 para a posição A ;

8º) colocar a constante 6 em B ;

9º) PUG DIV (calcula $\frac{x^3}{6}$) ;

10º) passar C $= \frac{x^3}{6}$ para a posição XTS, para uso posterior;

11º) passar X para A;

12º) passar C para B;

13º) PUG SUBT (calcula $x - \frac{x^3}{6}$) ;

14º) passar C $= x - \frac{x^3}{6}$ para TRM, para uso posterior;

15º) passar XDO $= x^2$ para A;

16º) passar XTS $= \frac{x^3}{6}$ para B;

17º) PUG MPY ;

18º) passar C $= \frac{x^5}{6}$ para A;

19º) colocar a constante 20 em B;

20º) PUG DIV ;

21º) passar C $= \frac{x^5}{5!}$ para B;

22º) passar TRM $= x - \frac{x^3}{6}$ para A;

23º) PUG ADD ;

24º) colocar C $= x - \frac{x^3}{3!} + \frac{x^5}{5!}$ em SIN (o resultado) ;

25º) fim da subrotina.

Evidentemente, os pontos de acesso ADD e SUBT de FLUTA e MPY e DIV de FLUTB, devem ser declarados externos na rotina que os está chamando.

As variáveis mencionadas são variáveis lógicas (isto é, da estrutura lógica de programa), mas fisicamente elas vão ocupar varias palavras da memória, pois representam números em ponto flutuante.

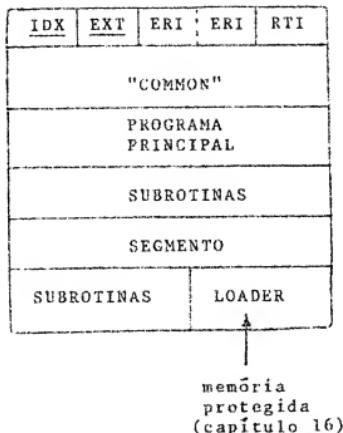
Vê-se claramente no exemplo acima como, mesmo programas simples, ficam grandes numa linguagem tão próxima à da máquina, e também como o programa quase nada faz a não ser chamar subrotinas. Isto sem contar as "sub-subrotinas" necessárias

para mover os dados de lugar (pois são constituídos de várias palavras) e para fazer FLUTA e FLUTB.

Nota-se que, já que tanto FLUTA como FLUTB mexem nas mesmas posições de memória que estão no início da área comum, é necessário ficar trocando de posição os valores calculados e novos argumentos de funções, para realizar a comunicação entre as subrotinas. Uma alternativa para isso é usar endereçamento indireto (capítulo 4 e 5) na subrotina e passar, em vez do próprio valor numérico da variável, o seu endereço. Para fazer isso, usa-se uma outra subrotina que já está pronta, chamada ENTR, cujas instruções de uso devem ser consultadas. Exemplo:

Programa Principal			Subrotina
variável	endereço	valor	variável
A	/425	/AB	M (duas palavras)

Passando o endereço de A para M, esta toma o valor /04 na primeira palavra e /25 na segunda palavra.
Se se tiver agora, na subrotina, as instruções:

```
    :
    IND      (endereçamento indireto)
    SOM   M
    :
```

será somado /AB, que é o conteúdo de A. Deste modo, não foi necessário colocar A na área comum. Este método é o geralmente empregado para passar poucos argumentos de uma rotina a outra. Para um número grande de argumentos, é preferível usar a área comum.

Divisão da Memória entre as várias rotinas; ligação

Quando se termina de realizar a montagem, dispõe-se de várias fitas-objeto, cada uma com uma rotina em formato relocável. A estas juntam-se, eventualmente, fitas provenientes da biblioteca de programas. Fica-se, então, com uma pilha de fitas contendo um programa principal, subrotinas e segmentos, com área comum reservada. Surge então a questão de como colocá-los na memória, principalmente se o programa completo não couber na mesma.

A alocação de memória que geralmente se usa é a seguinte:

IDX = indexador (posição 000)

EXT = extensão do acumulador (posição 001)

ERI E ERI = endereço de retorno de interrupção (posição 002 e 003)

RTI = início de uma rotina de tratamento de interrupção (se houver)

O comprimento da área comum é aquele declarado no programa principal. Por isso, nenhuma subrotina ou segmento pode ter área comum maior, porque senão essas posições avançariam para dentro do programa principal e destruiriam informações valiosas.

No início, deixa-se na memória, a área comum, o programa principal e as subrotinas chamadas por este, além das "sub-subrotinas" (aquelas chamadas pelas subrotinas). A seguir vem uma região reservada a um segmento e suas subrotinas, que conterá,de cada vez, um segmento — o que estiver sendo executado (pois supõe-se que mais segmentos não caberão na memória, simultaneamente). Naturalmente, só uma cópia de uma dada subrotina é necessária na memória em um instante, mesmo que seja chamada de diversos pontos em outras subrotinas e segmentos.

Ligação

Será necessário fazer a ligação entre as várias unidades do programa, ou seja, por exemplo, calcular os endereços absolutos de posições externas referenciadas em uma das unidades do programa, que dependerão dos locais físicos onde forem postas as rotinas onde esses símbolos foram definidos.

Para executar esse trabalho existe um programa, chamado relocador-ligador, que gera uma fita binária em formato absoluto, ou seja, já executável pelo Patinho Feio, a partir das fitas com os códigos relocáveis das subrotinas, programa-fonte e segmentos(vide também o capítulo 16).

Sistema Monitor de Disco

Como se observa, a introdução dos conceitos relativos à relocação de programas aumenta enormemente os recursos

de programação, embora no início seja mais complicado do que os programas absolutos.

Quando, no futuro, o Patinho Feio dispuser de um disco para armazenagem de dados, será possível, com os métodos de relocação, implementar um sistema monitor de disco, que se encarregará de armazenar os códigos relocáveis provenientes da montagem diretamente no disco e a seguir realizará, automaticamente, a relocação e ligação das rotinas do programa. Isto evitará o atual estágio intermediário onde os códigos relocáveis ficam armazenados em fita de papel perfurada.

15 - PSEUDO-INSTRUÇÕES PARA O MONTADOR RELOCÁVEL / ASSOCIAÇÃO DE TIPOS DE VARIÁVEIS A OPERANDOS DE INSTRUÇÕES DE REFERÊNCIA À MEMÓRIA

Neste capítulo serão discutidas as pseudo-instruções para o montador relocável. Note-se que não são novas instruções para a máquina, mas apenas para o montador (por isso são chamadas pseudo-instruções). Isto apenas reflete o fato de a relocação ser um conceito puramente da área de "software", como já foi dito no capítulo 14.

Todas as pseudo-instruções disponíveis no montador absoluto existem também no montador relocável, embora, às vezes, com significados um pouco diferentes. Além disso, existem ainda novas pseudo-instruções.

Começar-se-á vendo como se identificam os tipos de símbolos e constantes no montador relocável.

Como se sabe, a diferença entre programas absolutos e relocáveis aparece apenas quando há uma referência à memória (endereço), pois nos programas relocáveis a maioria dos endereços são relativos ao início da rotina, embora possa também haver endereços absolutos, além de pontos de acesso, endereço externo e variáveis na área comum.

Torna-se então necessária uma regra pela qual o montador relocável possa determinar de que espécie de endereço se está tratando. Para isso estabeleceu-se que <u>operandos</u> válidos são os seguintes (consultar o diagrama de precedências,no capítulo 5 ou no apêndice):

a) <símbolo> = referencia o endereço correspondente ao símbolo. O símbolo é do mesmo tipo que o endereço referido.Exemplo: se o endereço for relativo, o símbolo é relativo

b) <símbolo> <sinal> <deslocamento> = análogo ao anterior, mas ao endereço referenciado pelo símbolo soma-se ou subtrai-se (conforme o sinal) um deslocamento(dado em número de palavras).

c) <endereço> (constante) = é um endereço absoluto na memória.

d) * = é o endereço da 1a. palavra da própria instrução que tem esse operando. Portanto, é sempre relocável.

e) *-* = é uma outra forma de endereçar a posição absoluta zero da memória (indexador).

f) *<sinal> <deslocamento> = é sempre um endereço relocável , correspondente ao endereço da 1a. palavra da instrução com um deslocamento para mais ou para menos.

g) .<sinal>= é o endereço correspondente à instrução rotulada com . mais próxima da instrução atual para cima (se for .-) ou para baixo (se for .+) no programa. É sempre relocável.

h) .<sinal$_1$><sinal$_2$><deslocamento> = também relocável; é o endereço referenciado como no item anterior, acrescido ou diminuído (conforme o sinal$_2$) de um deslocamento (em número de palavras).

i) .<sinal> <hexadecimal> um dígito (=N) = refere-se ao endereço (sempre relocável) da instrução "distante de N pontos para cima (.-<N>) ou para baixo (.+<N>) da instrução atual. N é um dígito hexadecimal, sem / na frente.

j) .<$sinal_1$><N> <$sinal_2$> <deslocamento> = é o endereço relocável obtido a partir do anterior, aplicando-se um deslocamento para a frente ou para trás (de acordo com o $sinal_2$).

Exemplos:

1) Supondo que PTU referencie o endereço relocável /02A, A@@ o endereço absoluto /9A3, PPO seja externo, CTN tenha endereço /00A relativo ao início da área comum, e que a atual instrução do programa tenha sua primeira palavra no endereço relocável /10B, tem-se:

<operando>	endereço referido e tipo	
PTU	/02A	relocável
A	/9A3	absoluto
PPO	***	ext.(o endereço final dependerá da ordem em que as rotinas forem processadas pelo carregador absoluto)
CTN	/00A	comum
PTU+5	/02F	relocável
A-3	/9A0	absoluto
CTN+/F	/019	comum
/27B	/27B	absoluto
*	/10B	relocável

<operando>	endereço referido e tipo	
-	/000	absoluto
*+/F	/11A	relocável
*-10	/0FB	relocável

2) Endereço Relocável

```
0A1   .  CAR ....
          :
          :
1A2      PLA <operando>
          :
          :
1FF   .  SOMI /B
          :
          :
208   .  TRI
          :
          :
```

<operando>	endereço (todos relocáveis)
.-	/0A1
.+	/1FF
.-+/A	/0AB
.++12	/20B
.+2	/208
.+2-2	/206
.+2+/A0	/2A8

Rótulos ("labels")

Os rótulos, que servem para dar um nome a uma posição de memória, podem se constituir de 2 tipos: 1º) um símbolo; 2º) um ponto (.). As regras para se determinar a que tipo de endereço um dado rótulo faz referência são dadas pelas pseudo-instruções do montador relocável.

Pseudo-Instruções

(Obs.: não é mostrado a seguir, o campo de comentários das pseudo-instruções, que pode existir).

Na segunda linha de qualquer programa relocável (logo após a linha de controle — vide capítulo 16) é obrigatório declarar ao montador que espécie de programa está contido na fita-fonte. Para isto, usa-se uma das três pseudo-instruções seguintes:

1) b̸b̸ .... b̸NOMEb̸b̸ .... b̸<nome do programa> : se for programa principal.

2) b̸b̸ .... b̸SUBRb̸b̸ .... b̸<nome> : se for subrotina.

3) b̸b̸ .... b̸SEGMb̸b̸ .... b̸<nome> : se for segmento.

   b̸b̸ .... b̸ significa tantos espaços em branco quantos desejarmos, e está subentendido no que segue.

   Do nome declarado só se conservam as duas primeiras e a última letras. Não é permitido haver duas rotinas com o mesmo nome, nem usar o nome de uma rotina como um símbolo em qualquer outra parte do programa.

Outras pseudo-instruções(podem ser postas em qualquer parte do programa, exceto a pseudo-instrução FIM):

4) ORG <operando>

   Esta instrução tem aqui um significado diferente daquele do montador absoluto.

   <operando> refere-se obrigatoriamente a um endereço relativo já definido quando a pseudo-instrução ORG for encontrada.

   O efeito desta instrução é causar a numeração das instruções subsequentes a partir do endereço definido pelo operando , modificando, portanto, a numeração sequencial.

Exemplo:

```
       :
JAT   CAR   PLE
       :
      ORG   JAT + /B5
      TRI
       :
```

Se o endereço relativo correspondente a JAT for /100, a instrução TRI terá endereço relativo /1B5.

Esta instrução raramente é usada no montador relocável.

Pseudo-instruções para ligação entre rotinas:

5) EXT <símbolo>

Declara o símbolo como global externo, isto é, o símbolo é referenciado nesta rotina, mas definido em outra. Usa-se para chamada de subrotinas.

Exemplo:

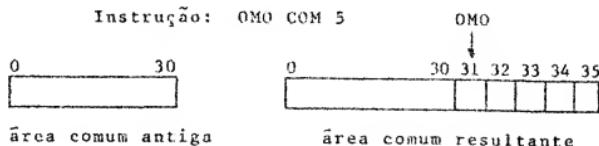
```
       :
EXT    WRITE
       :
PUG    WRITE
       :
```

Se WRITE não fosse declarado <u>externo</u>, ficaria sendo um símbolo indefinido.

6) ENT <operando>

Declara que o <operando> referencia um ponto de acesso da rotina em questão.

Exemplo:

```
        SUBR   IO
         :
        ENT    WRITE
         :
WRITE   PLA    *
         :
```

Quando, na rotina do item anterior, for executado o PUG WRITE, a execução da subrotina IO começará no respectivo ponto de acesso.

O operando deve referenciar um endereço <u>relativo</u> (relocável).

7) <símbolo> COM <tamanho>
(opcional)

Adiciona à área comum já definida nesta rotina, mais palavras, cujo número é dado pelo <tamanho>. A primeira dessas palavras passa a ter seu endereço referenciado pelo <símbolo> (se existir). (O endereço é em relação ao início da área comum).

Exemplo:

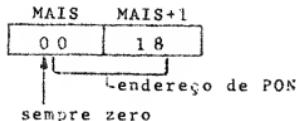
Instrução: OMO COM 5

Por exemplo, CAR OMO+2 carrega no <u>ACC</u> o conteúdo da posição 33 da área comum.

Naturalmente, se nenhuma área comum tiver ainda sido declarada, haverá a inicialização nesse instante.

8) <rótulo> DEFC <valor> (define constante)
(opcional)

Coloca o <u>valor</u> como conteúdo da palavra corrente da memória, que fica referenciada pelo <u>rótulo</u>, que, portanto, fica referenciando uma posição <u>relocável</u>.

9) <rótulo> DEFASC "<mensagem>" (define mensagem)
(opcional)

Define uma mensagem codificada em ASCII (vide apêndice) que deve estar entre aspas. É equivalente a uma sequência de DEFC, conforme o exemplo a seguir:

A <mensagem> deve ter, no máximo, 50 caracteres.

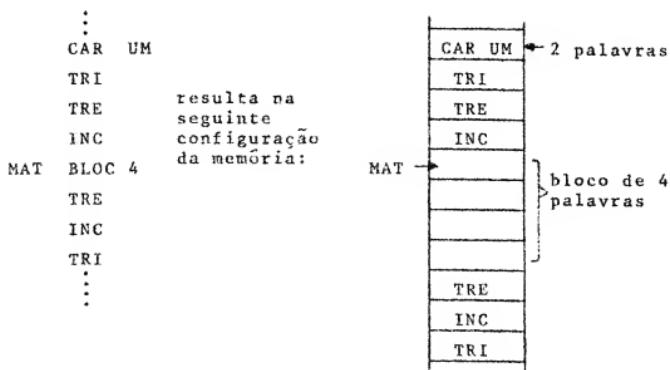
10) <rótulo> DEFE <operando> (define endereço)
(opcional)

Coloca na posição de memória corrente e na seguinte (duas palavras, portanto) o endereço referenciado pelo operando. Exemplo: supondo o endereço relativo de PON = /018, com a instrução MAIS DEFE PON, o montador colocará em MAIS e MAIS+1 os números /00 e 18:

Esta instrução é utilizada quando se estiver empregando endereçamento indireto (capítulo 4 e 5).

11) <rótulo> DEFI <operando> (define indireto) (opcional)

Análogo ao anterior, mas o primeiro dígito hexadecimal da primeira palavra é feito igual a um. É usado quando há endereçamento indireto em mais de um nível.

Exemplo:

MAIS DEFI PON resultaria em

MAIS	MAIS+1
10	18

Obs.: Note-se que DEFC envolve uma palavra da memória, enquanto DEFE e DEFI envolvem duas palavras cada uma.

12) <rótulo> BLOC <tamanho> (bloco de dados) (opcional)

Reserva uma área com o tamanho especificado de palavras para armazenagem de dados, sem colocar nenhum valor nessas palavras. O rótulo (se existir), fica associado à primeira palavra do bloco.

Exemplo:

Obs.: Note-se a diferença entre as pseudo-instruções BLOC e COM:

- BLOC reserva um grupo de palavras "internas" à rotina em questão.

- COM reserva um grupo de palavras numa área comum a todas as rotinas.

13) <símbolo> EQU <operando> (equivalência)

Faz com que o símbolo fique do mesmo tipo e com o mesmo endereço que aquele referenciado pelo operando. O operando deve ser tal que se refira a um endereço já definido quando a pseudo-instrução EQU for encontrada.
Exemplo:

PMU EQU /085 IDX EQU *-*	fazem PMU e IDX referenciarem os endereços absolutos /085 e /000.
QTX EQU QTY	faz QTX e QTY referirem-se à mesma posição de memória. Portanto, QTX e QTY ficam do mesmo tipo, isto é, se QTY for relocável, QTX também o será e, se QTY for absoluto, QTX também o será. QTY já deve ter sido definido anteriormente.

Esta instrução é usada para dar mais de um nome (rótulo) a uma posição de memória

14) FIM <operando>

É obrigatoriamente a última instrução de um programa. Serve para indicar ao montador relocável que acabaram as instruções a serem montadas, ou seja, acabou o programa-fonte.

O operando só tem significado num programa principal: neste caso ele referencia um endereço que é o endereço da posição de memória onde deve começar a execução do programa (isto é, a posição que contém a primeira instrução a ser executada). Por conseguinte, o endereço em questão será , normalmente, relocável.

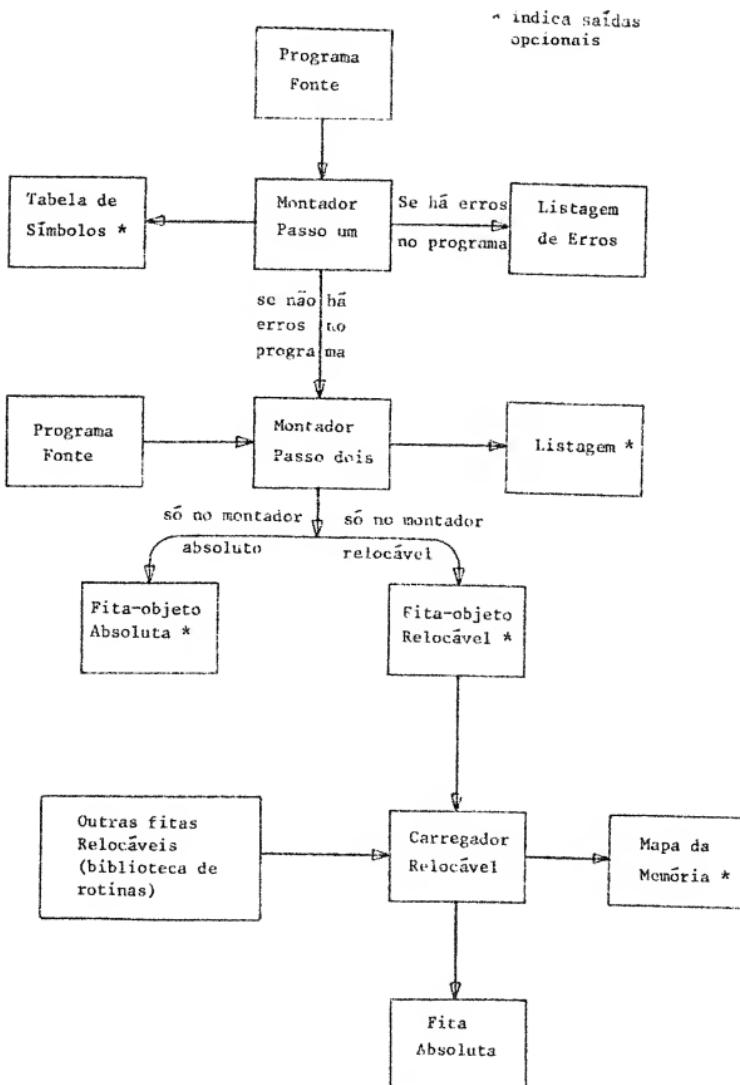
Se se tratar de uma subrotina ou segmento, o operando é ignorado, pois estas unidades do programa não podem ser iniciadas independentemente, mas apenas quando "chamadas".

Exemplo:

```
        :
        :
        FIM  STT  {return} {linefeed}
```

Para exemplos de programas relocáveis onde são usadas essas pseudo-instruções, consultar o apêndice.

16 - OPERAÇÃO DO PATINHO FEIO PARA MONTAGEM DE PROGRAMAS

Neste capítulo mostra-se como operar o Patinho Feio e realizar a montagem de um programa perfurado numa fita de papel (fita-fonte).

Devido ao fato de a memória do Patinho Feio ser de apenas 4K (4096) palavras, o montador não cabe inteiro nessa memória. Assim, tornou-se necessário dividir a montagem em 2 fases. No 1º passo, o montador lê, pela leitora de fita, a fita contendo o programa-fonte e monta na memória uma tabela de símbolos, onde estão todos os rótulos ("labels") definidos no programa; além disso, deteta eventuais erros no programa. Se, no fim do passo 1 não houver sido dada mensagem de erro, pode-se passar ao passo 2. Caso contrário, não adianta executar o passo 2, pois resultará uma fita-objeto errada.

No fim do passo 1 é impressa a tabela de símbolos. Se não houver nenhum erro, é impresso, a seguir, /00 SI (que significa: zero símbolos indefinidos) e PASSO 2, quando então o Patinho Feio está pronto para o passo 2.

No passo 2 a fita-fonte é novamente lida pelo montador, que agora, fazendo uso das tabelas montadas no passo 1, perfura uma fita-objeto e, simultaneamente, faz uma listagem do programa.

As três saídas (tabela de símbolos, listagem e fita-objeto) são opcionais e são determinadas pela primeira linha do programa-fonte, que é o controle do montador. Esta linha é obrigatória. Ela tem o seguinte formato:

1º) @ na coluna 1 (de modo que o primeiro caracter não branco encontrado pelo montador na fita de papel tem que ser uma @).

Esquema Geral de uma Montagem:

2º) Opcionalmente as letras B, L e T em qualquer ordem, significando:

B = quer-se fita-objeto
L = quer-se listagem
T = quer-se tabela de símbolos

3º) Para o montador relocável, pode-se ainda, opcionalmente, colocar o seguinte, após os caracteres B, L e T:

a) ,I para indicar que se trata de uma rotina de tratamento de interrupção (RTI); ou

b) ,Dn para indicar que se trata de um "driver" de E/S(rotina que trata da E/S) para o dispositivo n (n é um dígito hexadecimal).

4º) A seguir, RETURN e LINEFEED (nessa ordem) para indicar o fim da linha de controle.

Exemplos:

1) @T {RT} {LF} só é desejada a tabela de símbolos.

2) @TBL, DE {RT}{LF} quer-se tudo e é um "driver" de E/S para o dispositivo /E (só no montador-relocável).

3) @LB {RT}{LF} quer-se listagem e fita-objeto.

Observações:

1º) Se for detetado algum erro durante o passo 1, a linha onde ele ocorreu é impressa, em como o código do erro (ver código de erros no apêndice).

2º) Se houver símbolos indefinidos no programa, eles serão listados ao final do passo 1, mesmo que não tenha sido pedida a tabela de símbolos. A seguir, é impresso o número total de símbolos indefinidos; por exemplo: /0F SI.

3º) Ocorrendo algum dos casos anteriores, não será impressa a mensagem PASSO 2.

4º) Não se deve esquecer que após a instrução FIM <operando>, deve-se ter um RETURN e um LINEFEED para indicar o fim da linha, e esses serão os últimos caracteres perfurados na fita, nesta ordem.

Controle de listagem no montador relocável

No montador relocável é possível suprimir a listagem no meio de um programa, colocando um caracter "-" na coluna 1. Os códigos-objeto gerados, porém, continuam a ser listados. Para recomeçar a listagem coloca-se um "+" na coluna 1. As linhas começadas com "-" e "+" são também consideradas como linhas de comentário, além daquelas que começarem com um "*".

Explicação sobre o formato de diversas saídas

a) Tabela de símbolos no montador absoluto:

Saem duas colunas, a primeira com o nome do símbolo(três caracteres) e a segunda com o endereço absoluto correspondente na memória. Quando o símbolo tem menos de 3 caracteres, o espaço restante é preenchido com @ .

Exemplo:

NCS	100	endereço em hexadecimal
CCS	101	
N@@	105	
A@@	109	
PEF	1A8	
MAQ	---	←—— símbolo indefinido
PT@	12D	

b) Tabela de símbolos no montador relocável:

A tabela agora tem 3 colunas: na primeira vão os nomes dos símbolos, na segunda vão os endereços e na terceira vai o tipo de símbolo,que é: absoluto, relativo (neste caso, deixa-se em branco a terceira coluna), ponto de acesso, externo, comum.

Símbolo

- Absoluto : o endereço é absoluto na memória.
- Relativo(relocável): o endereço é relativo ao início do programa.
- Ponto de Acesso: o endereço é relativo ao início do programa.
- Externo : não há endereço por ser externo; na coluna de endereços são colocados três asteriscos.
- Comum : endereço relativo ao início da área comum.

Exemplo:

HEN	000	ENT
ACT	17A	ABS
T@@	025	
SAE	026	
COL	***	EXT
ENM	03B	ENT
CM@	01F	COM

c) Listagem do montador absoluto:

Formato de uma linha de listagem:

1º) número da linha (em decimal);

2º) endereço local, isto é, aquele referido pelo operando "*" numa instrução de referência à memória (é o endereço onde será armazenada a primeira palavra da instrução (em hexadecimal) ou de um rótulo);

3º) código-objeto gerado (em hexadecimal); uma ou duas palavras, conforme a instrução;

4º) rótulo,que pode ser um nome (símbolo) ou um "." ;

5º) mnemônico da instrução;

6º) operando (se existir) ou * se a instrução não exigir operando;

7º) comentários.

Se uma linha for comentário, ela somente é numerada e copiada. O mesmo ocorre para o controle do montador (1ª linha).

Na instrução FIM, o endereço do 2º campo é o de início da execução, conforme definido no operando.

Exemplo:

```
1  @TLB
2   100                  ORG  /100
3   100 00          NCS  DEFC /00    primeira variável
```

exemplo da folha de listagem

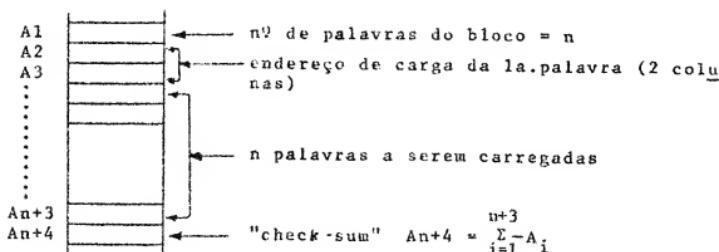
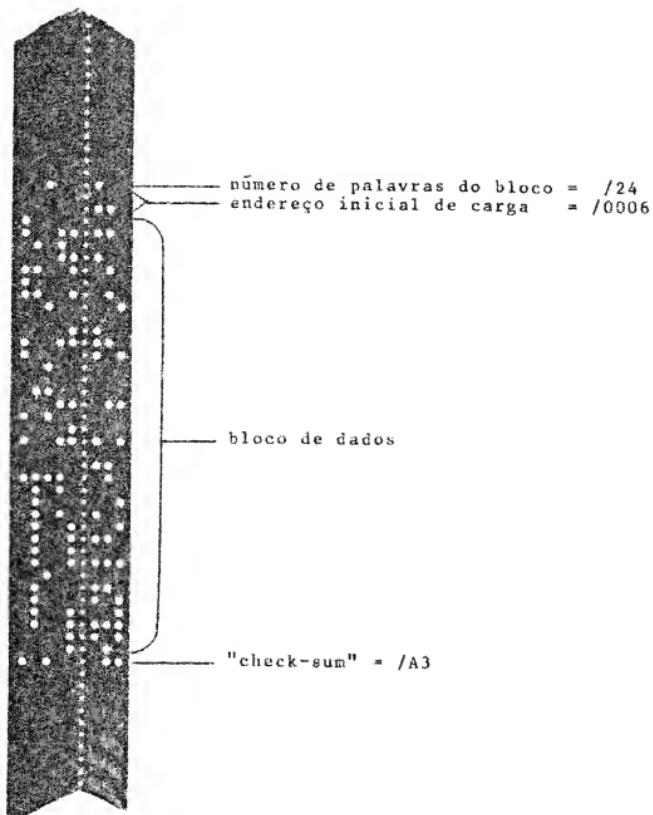
```
INO  006
LOP  008
FRE  01C
N@G  01B

/00 SI

#PASS02

   1   @BLT
   2   006                  ORG     6
   3   *
   4   ***************************************************************
   5   *                                                             *
   6   *  PROGRAMA-EXEMPLO ABSOLUTO PARA O MANUAL DO MONTADOR DO     *
   7   *  PATINHO FEIO: ESTE PROGRAMA ESCREVE "PATINHO FEIO" NA      *
   8   *  DECWRITER (ENDERECO DE E/S /A).                            *
   9   *                                                             *
  10   ***************************************************************
  11   *
  12   006 80      INICIO  LIMPO   *        APONTA PARA A PRIMEIRA
  13   007 9E              TRI     *        LETRA DA FRASE.
  14   008 50 1C   LOOP    CARX    FRASE    BUSCA A PROXIMA LETRA A ESCREVER.
  15   00A CA 80           SAI     /A0      ESCREVE NA DECWRITER.
  16   00C CA 21           SAL     /A1      LOOP DE
  17   00E 00 0C           PLA     *-2      "WAIT-FOR-FLAG".
  18   010 9E              TRI     *        FAZ O INDEXADOR
  19   011 85              INC     *        APONTAR O PROXIMO
  20   012 20 00           ARM     0        CARACTER A SER ESCRITO.
  21   014 60 1B           SOM     N        TESTA SE A FRASE TERMINOU.
  22   016 A0 08           PLAN    LOOP     NAO: VAI ESCREVER MAIS;
  23   018 9D              PARE    *        SIM: PARA.
  24   019 00 06           PLA     INICIO   SE FOR DADA NOVA "PARTIDA",
  25   *                     RECOMECA O PROGRAMA.
  26   01B F2      N       DEFC    -14      NUMERO DE CARACTERES DA FRASE (INCLU.
  27   *                     RETURN E LINEFEED) COM SINAL TROCADO.
  28   01C 50      FRASE   DEFC    @P
  29   01D 41              DEFC    @A
  30   01E 54              DEFC    @T       TABELA COM A
  31   01F 49              DEFC    @I       FRASE A SER
  32   020 4E              DEFC    @N       ESCRITA.
  33   021 48              DEFC    @H
  34   022 4F              DEFC    @O
  35   023 20              DEFC    @
  36   024 46              DEFC    @F
  37   025 45              DEFC    @E
  38   026 49              DEFC    @I
  39   027 4F              DEFC    @O
  40   028 0D              DEFC    /0D      CODIGO DE RETURN.
  41   029 0A              DEFC    /0A      CODIGO DE LINEFEED.
  42   *
################################  43   006                 FIM     INICIO
```

d) Listagem do montador relocável:

É semelhante à do montador absoluto, com duas particularidades:

1) o endereço listado no 2º campo é relativo ao início do programa (a não ser quando se tratar de rótulos absolutos);

2) após o código-objeto hexadecimal, nas instruções que referenciam a memória, vem o tipo de código:

R = relocável (endereço relativo ao início do programa)
X = instrução indexada
em branco = instrução que não referencia a memória ou absoluta.

e) Fita-objeto do montador absoluto:

A fita-objeto do montador absoluto já está em formato binário executável, isto é, o programa nela perfurado pode ser carregado na memória e executado imediatamente.
Os dados vem perfurados na fita, em <u>blocos</u>, cada um com o seguinte formato:

O "check-sum" testa se houve erro na perfuração, o que terá ocorrido se a soma de <u>todas</u> as palavras do bloco ($\sum_{1}^{n+4} A_i$) não for zero.

Fita-objeto absoluta correspondente ao programa da página 16.7:

Após o último bloco deve haver, pelo menos, cerca de 30 "feed-frames" (nulos) para indicar o fim da fita.

f) Fita-objeto do montador relocável:

Não é, ainda, executável, pois está num formato binário intermediário. Para tornar-se executável, é necessário passá-la por um outro programa (relocador-ligador) que faça a ligação entre os diversos programas e subrotinas que então, dará como saída, uma fita com código binário executável.

Devido ao maior número de informações que uma fita-objeto relocável deve conter, seu formato é mais complicado que aquele da fita-objeto do montador absoluto. Contém o seguinte:

1) um bloco de início - contém o tipo do programa (principal, subrotina(normal,RTI,"Driver") ou segmento), o seu nome e o tamanho da área comum;

2) um bloco de ENT - contém os nomes e respectivos endereços dos símbolos declarados como pontos de acesso do programa;

3) um bloco de EXT - contém os nomes dos símbolos declarados externos;

4) blocos de dados - contém o código-objeto relocável (além da instrução, vem também a informação sobre o seu tipo);

5) um bloco de FIM - contém o endereço de execução (se for necessário).

Exemplos de programas completos, absolutos e relocáveis, são encontrados no apêndice.

Operação do Patinho Feio para realizar uma montagem (Consulte também o "Manual de Operação do Patinho Feio")

Carregador Absoluto:

Todo programa, para ser executado, deve ser colocado na memória do computador. Isto pode ser feito, por exemplo, através do Registrador de Chaves do Painel. Esse método é, porém, extremamente trabalhoso e sujeito a erros.

Por isso, foi desenvolvido um programa, chamado Carregador ("loader") absoluto, cuja única função é transferir para a memória, um programa perfurado em uma fita de papel, no mesmo formato que aquele gerado pelo montador absoluto(vide página 16.8). Este programa carregador, para ser executado deve, naturalmente, estar na memória. Para evitar a colocação deste programa na memória pelo Registrador de Chaves, sempre que o Patinho Feio fosse ligado, criou-se uma área protegida na memória do Patinho Feio, que não é destruída ao desligar-se o computador, e nesta área foi colocado o carregador absoluto.

Além disso, normalmente, nada pode ser gravado ou lido nesta área, e consequentemente, o programa nela armazenado não pode ser executado. Desta forma, a área está protegida contra eventuais "acidentes".

Para desproteger está área existe um botão no painel, e desta forma pode-se processar o programa carregador e carregar com ele outros programas na memória. Deve-se tomar muito cuidado quando a memória estiver desprotegida, para não estragar seu conteúdo; caso contrário, dever-se-á novamente colocá-lo na memória, usando para isto o "micro-pré-carregador". Instruções de como fazer isso estão afixadas no próprio painel do Patinho Feio.

A área protegida começa na posição /F80 e vai até o fim da memória (/FFF).

Como realizar uma montagem:

a) Ligar o Patinho Feio e a leitora de fita e colocar a fita com o montador (passo 1) na leitora.

b) Apertar o botão "preparação".

c) Apertar o botão "endereçamento".

d) Colocar no Registrador de Chaves o número hexadecimal /F80.

e) Apertar o botão "partida" (conferir, no CI, se o número colocado é mesmo /F80).

f) Desproteger a memória.

g) Apertar o botão "normal".

h) Apertar o botão "partida". Neste ponto o passo 1 do montador é colocado na memória, após o que o Patinho Feio pára. Se parar com ACC = 0 está tudo O.K., caso contrário, voltar ao item c. (Houve erro de "check-sum").

i) Proteger a memória.

j) Colocar a fita com o programa-fonte na leitora de fita e ligar o dispositivo de listagem (Impressora ou DECwriter).

l) Apertar o botão "partida". O programa-fonte é lido e eventuais erros são detetados. A seguir é impressa a tabela de símbolos (se foi pedida) e, se não houve erro, é impresso:

```
/00 SI
PASSO 2
```

e então pode-se passar ao passo 2 do montador.

Caso seja necessário reiniciar o processamento do passo 1 , cujo endereço de execução é /006, deve-se, após recolocar a fita com o programa-fonte na leitora de fita:

1º) apertar "endereçamento" e "preparação";

2º) colocar /006 no RC e apertar "partida";

3º) apertar "normal" e "partida".

m) Se não houve erros no passo 1, carregar o passo 2 do montador na memória, da mesma forma como foi carregado o passo 1 (itens a, c, d, e, f, g, h, i, j).

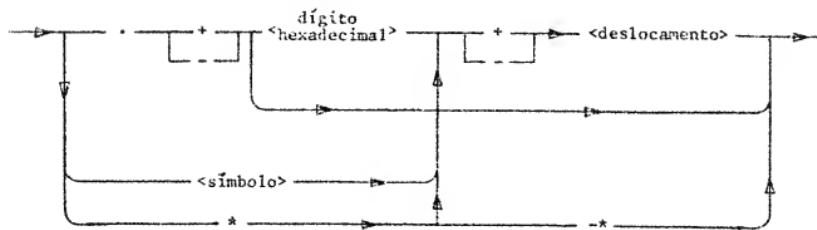
n) Recolocar a fita com o programa-fonte na leitora de fita.

o) ligar a perfuradora de fita, se foi pedida fita-objeto.

p) apertar "partida" . O passo 2 é executado (a partir do endereço /006) e dá a listagem (se foi pedida) e a fita-objeto (se foi pedida).

A fita-objeto resultante de uma montagem absoluta pode ser a seguir carregada na memória (da mesma forma como foi carregado o montador) e executada.

Já a fita que resulta do montador relocável não está ainda em formato executável, conforme já foi dito. Para isso, é necessário passá-la pelo relocador-ligador onde será feita sua ligação com outros segmentos ou subrotinas.

Observação:

O botão preparação ("reset") serve para colocar o Patinho Feio num estado conhecido: ele pára imediatamente (sem terminar a instrução que estiver executando) e vai para o modo endereçamento, colocando os vários flip-flops nos valores descritos no capítulo 12.

Além disso, apertar preparação é o único modo de escapar de um "loop" de endereçamento indireto (capítulo 5).

Apertar preparação durante a execução de um programa pode estragar o conteúdo da memória, devendo, portanto, ser evitada esta atitude.

APÊNDICE

a) Diagrama de precedências para construção de operandos válidos em instruções de referência à memória:

<símbolo> é uma sequência de uma a sete letras, das quais são retidas pelo montador apenas as duas primeiras e a última.

<dígito hexadecimal> dá o deslocamento em número de pontos.

<deslocamento> dá o deslocamento em número de palavras da memória.

b) Instruções do Patinho Feio:

INSTRUÇÕES DE REFERÊNCIA À MEMÓRIA

Código de Máquina	Mnemônico	Operando	Descrição Sucinta	Função
0nnn	PLA		CI ← eee	pulo incondicional
1nnn	PLAX		CI ← eee	pulo incondicional indexado.
2nnn	ARM		(eee) ← ACC	armazena
3nnn	ARMX		(eee) ← ACC	armazena indexado
4nnn	CAR	VER APÊNDICE a	ACC ← (eee)	carrega
5nnn	CARX		ACC ← (eee)	carrega indexado
6nnn	SOM		ACC ← ACC + (eee)	soma
7nnn	SOMX		ACC ← ACC + (eee)	soma indexado
Annn	PLAN		CI ← eee se ACC < 0	pula se negativo
Bnnn	PLAZ		CI ← eee se ACC = 0	pula se zero
Ennn	SUS		se (eee) ≠ 0 → (eee) ← (eee) − 1 se (eee) = 0 → CI ← CI + 2	subtrai um ou salta
Fnnn	PUG		em eee e (eee+1), monta a instrução PLA CI e faz CI ← eee+2	pula para subrotina (pula e guarda)

Na tabela acima, eee = endereço efetivo; nnn é calculado a partir do operando (eee e nnn são números hexadecimais).

INSTRUÇÕES DE ENTRADA E SAÍDA

Código	Mnemônico	Operando	t	Operação
Cn1t	FNC	/nt	t=0	desliga flip-flop que PERMITE/IMPEDE interrupção do dispositivo
			t=1	desliga flip-flop de ESTADO do dispositivo
			t=2	liga flip-flop de ESTADO do dispositivo
			t=4	desliga flip-flop de PEDIDO de interrupção do dispositivo
			t=5	liga flip-flop que PERMITE/IMPEDE interrupção do dispositivo
			t=6	liga flip-flop de CONTROLE e desliga flip-flop de ESTADO do dispositivo
			t=7	desliga flip-flop de CONTROLE do dispositivo
			t=8	ignora "feed-frames" (bytes nulos) (só para n = E)
Cn2t	SAL	/nt	t=1	$\underline{CI} \leftarrow \underline{CI} + 2$, se o flip-flop de ESTADO estiver ligado
			t=2	idem , se o dispositivo estiver O.K.
			t=4	idem , se o flip-flop de PEDIDO de interrupção estiver desligado.
Cn4t	ENTR	/nt	t=0	$\underline{ACC} \leftarrow$ registrador de oito bits do dispositivo
Cn8t	SAI	/nt	t=0	registrador de oito bits do dispositivo $\leftarrow \underline{ACC}$; desliga flip-flop de ESTADO e liga flip-flop de CONTROLE do dispositivo.

n = número do dispositivo; t = tipo de operação

INSTRUÇÕES DE DESLOCAMENTO

D1 0n	DD	n	0 → a→a→a→a→a→a→a→a→v → perdido	deslocamento à direita
D1 1n	DDV	n	a→a→a→a→a→a→a→a→v (v retorna ao início)	deslocamento à direita, usando o registro V
D1 2n	GD	n	a→a→a→a→a→a→a→a→v → perdido (último a retorna ao início)	giro à direita
D1 3n	GDV	n	equivalente a DDV	equivalente a DDV
D1 4n	DE	n	perdido ← v←a←a←a←a←a←a←a←a ← 0	deslocamento à esquerda
D1 5n	DEV	n	v←a←a←a←a←a←a←a←a (v retorna ao fim)	idem, usando o registro V
D1 6n	GE	n	perdido ← v←a←a←a←a←a←a←a←a (primeiro a retorna ao fim)	giro à esquerda
D1 7n	GEV	n	equivalente a DEV	equivalente a DEV
D1 8n	DDS	n	s → a→a→a→a→a→a→a → v → perdido (s duplicado)	deslocamento à direita, com duplicação de sinal.

$0 \leqslant n \leqslant 4$

INSTRUÇÕES IMEDIATAS

Código	Mnemônico	Operando	Operação	Descrição
D2 mn	XOR	OPERANDO=CONSTANTE	ACC ← ACC XOR mn	"exclusive or" imediato
D4 mn	NAND		ACC ← ACC NAND mn	"nand" imediato
D8 mn	SOMI		ACC ← ACC + mn	soma imediata
DA mn	CARI		ACC ← mn	carrega imediato

mn é um número hexadecimal calculado a partir do operando.

GRUPO 1 DE INSTRUÇÕES CURTAS

Código	Mnemônico	Operando	Operação
80	LIMPO	NÃO TEM OPERANDO	ACC ← 0 , V ← 0 , T ← 0
81	UM		ACC ← 1 , V ← 0 , T ← 0
82	CMP1		ACC ← complemento de um do ACC , V ← 0 , T ← 0
83	CMP2		ACC ← complemento de dois do ACC , atualiza V e T
84	LIM		V ← 0 , T ← 0
85	INC		ACC ← ACC + 1 , atualiza V e T
86	UNEG		ACC ← -1, V ← 0 , T ← 0
87	LIMP1		ACC ← 0 , V ← 1 , T ← 0

INSTRUÇÕES DE PAINEL

Código	Instrução		Operação	Observação
88	PNL	0	ACC ← RC , V ← 0 , T ← 0	
89	PNL	1	ACC ← RC + 1	alteram V e T
8A	PNL	2	ACC ← RC - ACC - 1	alteram V e T
8B	PNL	3	ACC ← RC - ACC	alteram V e T
8C	PNL	4	ACC ← RC + ACC	alteram V e T
8D	PNL	5	ACC ← RC + ACC + 1	alteram V e T
8E	PNL	6	ACC ← RC - 1	alteram V e T
8F	PNL	7	ACC ← RC , V ← 1 , T ← 0	

RC é o registrador de chaves incompleto (os oito bits menos significativos)

CRUPO 2 DE INSTRUÇÕES CURTAS

90	ST	0	se T = 0, CI ← CI + 2
91	STM	0	se T = 0, CI ← CI + 2 e T = 1
92	ST	1	se T = 1, CI ← CI + 2
93	STM	1	se T = 1, CI ← CI + 2 e T = 0
94	SV	0	se V = 0, CI ← CI + 2
95	SVM	0	se V = 0, CI ← CI + 2 e V = 1
96	SV	1	se V = 1, CI ← CI + 2
97	SVM	1	se V = 1, CI ← CI + 2 e V = 0

GRUPO 3 DE INSTRUÇÕES CURTAS

Código	Mnemônico		Descrição
98	PUL		pula para /002 e liga flip-flop NÃO ESTÁ/ESTÁ em interrupção (termina a interrupção)
99	TRE	NÃO TEM OPER.NDO	ACC ↔ {EXT} troca os conteúdos do ACC e da EXT
9A	INIB		desliga flip-flop que PERMITE/INIBE interrupção do sistema
9B	PERM		liga flip-flop que PERMITE/INIBE interrupção do sistema
9C	ESP		pára o processamento até ser acionado o botão de partida ou ser aceito um pedido de interrupção
9D	PARE		pára o processamento até ser acionado o botão de partida
9E	TRI		ACC ↔ {IDX} troca os conteúdos do ACC e do IDX
9F	IND		liga o bit de endereçamento indireto (BEI)

c) Pseudo-Instruções (comandos para o montador):

MNEMÔNICO	OPERANDO	DESCRIÇÃO	OBSERVAÇÃO
ORG	qualquer referência à memória já definida.	dá uma origem a partir da qual as instruções subsequentes devem ser armazenadas.	em programas relocáveis o operando deve ser do tipo relocável; não admite rótulo.
NOME	um símbolo	especifica um programa principal.	só no montador relocável; não admite rótulo.
SUBR	idem	especifica uma subrotina	idem
SEGM	idem	especifica um segmento	idem
EXT	idem	declara que o símbolo é global externo("external")	idem
ENT	um símbolo relocável, com ou sem deslocamento.	dá a posição de um ponto de acesso ("entry point") da rotina.	idem
DEFC	uma constante	define constante	usa uma palavra da memória.
DEFASC	n caracteres entre aspas (n < 50)	define mensagem	usa n palavras da memória; é equivalente a n DEFC consecutivos; só no montador relocável.

(Cont.)

MNEMÔNICO	OPERANDO	DESCRIÇÃO	OBSERVAÇÃO
DEFE	qualquer referência à memória.	define endereço referenciado pelo operando.	ocupa duas palavras da memória.
DEFI	idem	define endereço indireto a partir do operando.	idem
BLOC	uma constante	reserva na memória o número de palavras especificado no operando.	não zera as palavras.
COM	uma constante	reserva na área comum o número de palavras especificado no operando.	não zera as palavras; só no montador relocável.
EQU	qualquer referência à memória, previamente definida.	dá ao endereço especificado no operando o nome que aparece como rótulo.	o rótulo é obrigatório.
FIM	idem	indica o fim do programa-fonte.	deve ser a última instrução do programa; em sub-rotinas e segmentos o operando é ignorado.

d) Diagrama lógico dos pedidos de interrupção:

(No diagrama a seguir, só está representado um dispositivo de E/S).

Modos de ligar os flip-flops:

1) PEDIDO - é ligado automaticamente quando o flip-flop de ESTADO estiver ligado.
   - é desligado pela instrução FNC /n4.

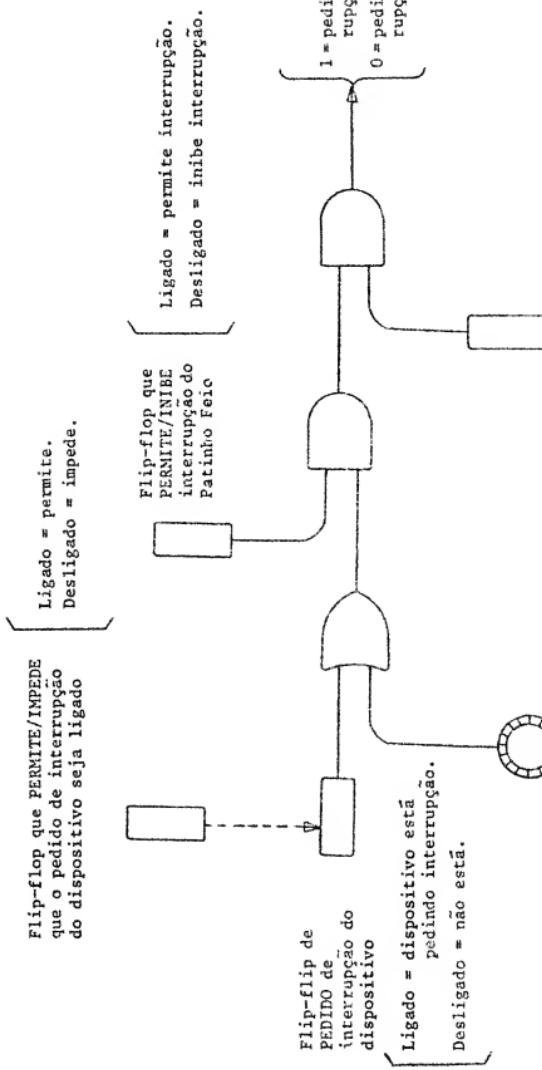
2) PERMITE/IMPEDE - é ligado pela instrução FNC /n5.
   é desligado pela instrução FNC /n0.

3) INTERRUPÇÃO - é ligado apertando-se o botão INTERRUPÇÃO do painel.
   é desligado pelo Patinho Feio, ao aceitar uma interrupção proveniente do painel.

4) PERMITE/INIBE - é ligado pela instrução PERM.
   é desligado pela instrução INIB.

5) NÃO ESTÁ/ESTÁ - é ligado pelc Patinho Feio,ao encerrar a interrupçãc (instrução PUL).
   é desligado elo Patinho Feio, ao aceitar uma interrupção.

Ao apertar-se o botão "preparação", os flip-flops tomam os seguintes valores:

PEDIDO: desligado

PERMITE/IMPEDE: desligado (impede)

INTERRUPÇÃO: desligado

PERMITE/INIBE: ligado (permite)

NÃO ESTÁ/ESTÁ: ligado (não está)

Flip-flop que PERMITE/IMPEDE que o pedido de interrupção do dispositivo seja ligado
Ligado = permite.
Desligado = impede.
Flip-flop que PERMITE/INIBE interrupção do Patinho Feio
Ligado = permite interrupção.
Desligado = inibe interrupção.
Flip-flip de PEDIDO de interrupção do dispositivo
Ligado = dispositivo está pedindo interrupção.
Desligado = não está.
Botão de INTERRUPÇÃO do painel
1 = pedido de interrupção aceito
0 = pedido de interrupção rejeitado
Flip-flop que diz se o Patinho Feio NÃO ESTÁ/ESTÁ interrompido
Ligado = não está interrompido.
Desligado = está interrompido.

e) Erros detetados pelo montador:

Após cada erro é apresentada, se necessário, uma lista de algumas das causas possíveis do erro.

ERRO 00 - OPERANDO DA PSEUDO "DEFASC" INCORRETO

Provavelmente falta fechar aspas após a mensagem que é o operando de uma DEFASC,ou a mensagem tem mais de 50 caracteres.

ERRO 01 - CONSTANTE HEXADECIMAL INVÁLIDA

Possíveis causas:

1) o sinal do número foi posto depois da "/"(deve ser antes).
2) foi usada uma letra que não é dígito hexadecimal (só são válidas letras de A a F).

ERRO 02 - ESTOURO DE MEMÓRIA

O endereço da memória ultrapassou /F7F.

Possíveis causas:

1) origem incorreta (muito alta).
2) deslocamento muito grande em operando de instrução de referência à memória.
3) programa muito grande, não cabe na memória: deve-se segmentá-lo.

ERRO 03 - ESTOURO DA TABELA DE SÍMBOLOS

São permitidos no máximo 256 símbolos em cada unidade do programa. Deve-se substituir alguns símbolos por endereçamento relativo (*± deslocamento), local (pontos ou absoluto, se possível).

ERRO 04 - COMANDO IRRECONHECÍVEL

Provavelmente há um erro no mnemônico ou falta o espaço entre rótulo e mnemônico e/ou entre mnemônico e operando.

ERRO 05 - CONSTANTE DECIMAL INVÁLIDA

ERRO 06 - OPERANDO INVÁLIDO

Possíveis causas:

1) NOME, SUBR, SEGM ou EXT com operando que não é um símbolo puro no montador relocável.
2) instrução que não é de referência à memória, com operando que não é uma constante.
3) instrução de deslocamento com operando não compreendido entre 0 e 4.
4) instrução de painel com operando não compreendido entre 0 e 7.

ERRO 07 - ORG, EQU OU FIM COM OPERANDO INVÁLIDO

Possíveis causas:

1) o operando não permite o cálculo do endereço a partir de elementos previamente definidos.
2) no montador relocável o operando de uma ORG deve ser do tipo relocável.

ERRO 08 - RÓTULO INVÁLIDO

Possíveis causas:

1) deslocamento no rótulo não é permitido (só no operando)
2) não há espaço entre rótulo e mnemônico, fazendo com que o rótulo fique com mais de sete letras.
3) rótulo com mais de sete letras.

ERRO 09 - SÍMBOLO DUPLAMENTE DEFINIDO

Cada símbolo pode aparecer como rótulo em apenas uma linha do programa.

ERRO 10 - OPERANDO INEXISTENTE

Não há operando em uma instrução que exige operando.

ERRO 11 - O PRIMEIRO COMANDO NÃO É NOME, SUBR OU SEGM

É obrigatório declarar o tipo da rotina (programa principal, subrotina ou segmento) logo após a linha de controle, no montador relocável.

ERRO 12 - OPERANDO DE PSEUDO INDEFINIDO

ERRO 13 - RÓTULO TIPO PONTO INDEFINIDO

ERRO 14 - RÓTULO DE PSEUDO INVÁLIDO

Possível causa: ORG, NOME, SUBR, SEGM, EXT, ENT, não admitem rótulo.

ERRO 15 - OPERANDO DA PSEUDO EXT COM DESLOCAMENTO

EXT exige como operando um símbolo puro.

ERRO 16 - PSEUDO PROIBIDA

Possíveis causas:

1) NOME, SUBR, SEGM ou FIM aparecem no meio do programa.

2) pseudo-instruções do montador relocável usadas no montador absoluto.

ERRO 17 - ESTOURO DA TABELA DE PONTOS

Mais de 256 linhas estão rotuladas com um ponto (.). Deve-se substituir alguns deles por endereçamento com símbolos, asteriscos ou absolutos, se possível.

ERRO 18 - PONTO INCORRETO

Possível causa: um operando com ponto não obedece as regras do diagrama de precedências (vide apêndice a).

ERRO 19 - FALTA ENDEREÇO DE EXECUÇÃO

A pseudo-instrução FIM de um programa principal está sem operando.

ERRO 20 - ERRO NA PRIMEIRA LINHA (COM @)

A linha de controle deve começar com @ , seguido das opções desejadas (B, L, T) sem espaços entre elas. No montador relocável pode-se por ainda",I" ou ",Dn" (vide capítulo 16).

f) Código ASCII:

ASCII = American Standard Code for Information Interchange.

É um código de sete bits por caracter. Contudo, costuma-se usar oito bits por caracter, onde o bit mais significativo é feito ou sempre zero ou um bit de paridade.

bits mais significativos		0	1	2	3	4	5	6	7
bits menos significativos		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	ƀ	Ø	@	P	'	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	:
D	1101	CR	GC	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑ ou ^	n	~
F	1111	SI	US	/	?	O	← ou _	o	DEL

g) Exemplo de Programa Absoluto:

```
HEM  E00
LER  E01
LEV  E7C
PRM  E34
SAI  E72
LIA  E23
LEX  E25
GUA  E31
PRE  E4F
UNC  E49
CAI  E4D
EHH  E5F
VAT  E65
VFF  E76
IGR  E7E
ACC  EC4
LOP  E86
DIS  EC5
ACH  EB2
BRO  EB5
ARH  EB6

/00 6I

BPASS02

   1  0BLTC
   2  E00                    ORG     /E00
   3  *
   4  * HEXAM - PROGRAMA QUE CARREGA A MEMORIA
   5  *         A PARTIR DE DADOS FORNECIDOS
   6  *         EM HEXADECIMAL PELA CONSOLE
   7  *
   8  ************************************************************************
   9  *
  10  *   HEXAM - INSTRUCOES DE UTILIZACAO:
  11  *
  12  * 1. ENDERECAR HEXAM
  13  * 2. DAR PARTIDA
  14  * 3. O CANAL 8 VAI FICAR ESPERANDO ENDERECAMENTO.
  15  * 4. PARA ENDERECAR A QUALQUER MOMENTO, BATER ARROBA (@).
  16  * 5. O COMPUTADOR RESPONDE C/ RETURN, 2 LINEFEEDS
  17  * 6. ENTRAR C/ ENDERECO EM HEXA, COM 3 DIGITOS.
  18  * 7. SE ERRAR, BASTA VOLTAR P/ 4 OU BATER UM BRANCO.
  19  *    NESTE CASO, O PROGRAMA IGNORA A ENTRADA ANTERIOR
  20  *    E AGUARDA NOVO ENDERECO.
  21  * 8. UMA VEZ ENDERECADO, OS DADOS QUE FOREM FORNECIDOS
  22  *    SERAO GUARDADOS EM SEQUENCIA A PARTIR DO ENDERECO
  23  *    ESPECIFICADO.
  24  * 9. OS DADOS DEVERAO VIR SEPARADOS POR UM UNICO BRANCO.
```

```
25   *10. O ULTIMO DADO DA LINHA NAO DEVE SER SEGUIDO DE BRANCO,
26   *    SENDO QUE NESTE CASO UM LINEFEED, RETURN OU VICE
27   *    VERSA O SUBSTITUIRA'
28   *11. UM BRANCO OU RETURN DEPOIS DO DADO E' UMA ORDEM P/
29   *    QUE O DADO SEJA ARMAZENADO.
30   *12. DEPOIS DE CADA BRANCO OU RETURN O BUFFER E' ZERADO.
31   *    E PORTANTO SE FOREM DADOS 2 BRANCOS EM SEQUENCIA
```

```
32   *    SERA' GUARDADO UM ZERO NO LUGAR DO SEGUNDO BRANCO.
33   *13. EM CASO DE ERRO NOS DADOS, SE O CARACTER FORNECIDO
34   *    FOR HEXADECIMAL,BASTA BATER DE NOVO EM SEGUIDA, SEM
35   *    BRANCOS, O DADO CORRETO. SO' SAO GUARDADOS NA MEMORIA
36   *    OS DOIS ULTIMOS DIGITOS.
37   *14. SE O CARATER NAO FOR HEXADECIMAL, O COMPUTADOR RESPONDE
38   *    COM UMA SETA (_) E PARA O PROCESSAMENTO.
39   *15. NESTE CASO, DANDO PARTIDA, O PROGRAMA VOLTA A SER
40   *    EXECUTADO COMO NO CASO 14.
41   *16. ANTES DE DAR ENDERECAMENTO, E' PRECISO NAO ESQUECER
42   *    DE GUARDAR O DADO ANTERIOR. SE NAO FOR DADO UM BRANCO
43   *    OU RETURN, O DADO NAO SERA' ARMAZENADO.
44   *
45   ************************************************************************
46   *
47   E00 9A      HEXAM   INIB     *        INIBE INTERRUPCAO
48   *
49   * SECAO DE LEITURA DE ENDERECO
50   *
51   E01 FE 7C  LEENDER PUG      LECONV   LE PRIMEIRO CAR. DO END.
52   E03 AE 01          PLAN     LEENDER  SE BCO. OU RETURN, VOLTA
53   E05 D2 20          XOR      /20      NAO: MONTA "ARM"
54   E07 2E 34          ARM      ARM      GUARDA P/EXECUTAR
55   E09 FE 7C          PUG      LECONV   LE SEG. CARATER
56   E0B AE 01          PLAN     LEENDER  SE BCO, VOLTA A LER ENDERECO
57   E0D D1 4F          DE       4        AJEITA P/ COMPOR
58   E0F 2E 35          ARM      ARM+1    GUARDA
59   E11 FE 7C          PUG      LECONV   LE TERCEIRO CARATER
60   E13 AE 01          PLAN     LEENDER  SE BCO, VOLTA A LER END.
61   E15 6E 35          SOM      ARM+1    SE NAO COMPOE COM SEGUNDO DIG.
62   E17 2E 35          ARM      ARM+1    GUARDA P/ EXECUTAR
63   E19 DA 0D          CARI     /0D      SAI RETURN
64   E1B FE 72          PUG      SAI      NA TTY
65   E1D DA 0A          CARI     /0A      SAI LINEFEED
66   E1F FE 72          PUG      SAI      NA TTY
67   E21 FE 72          PUG      SAI      IDEM
68   *
69   * LEITURA DE UMA PALAVRA
```

```
 70   *
 71   E23 80       LIMPA   LIMPO    *        ZERA
 72   E24 99               TRE      *        EXTENSAO
 73   E25 FE 7C    LEPROX  PUG      LECONV   LE UM CARATER
 74   E27 AE 31            PLAN     GUARDA   SE BCO OU LINEFEED, STORE.
 75   E29 99               TRE      *        SE NAO, TRAZ EXTENSAO
 76   E2A D1 4F            DE       4        AJEITA P/ COMPOR
 77   E2C 60 01            SOM      /001     COMPOE
 78   E2E 99               TRE      *        C/ DIGITO LIDO
 79   E2F 0E 25            PLA      LEPROX   CONTINUA LENDO ATE' ACHAR BCO/RET
 80   *
 81   * ARMAZENAMENTO NA MEMORIA
 82   *
 83   E31 FE 4F    GUARDA  PUG      PROTEGE  TESTA SE ENDERECO INVADE
 84   E33 99               TRE      *        HEXAM. SE NAO,
 85   E34 20 00    ARM     ARM      *-*      GUARDA EXTENSAO NO ENDERECO CONT.
 86   E36 4E 35            CAR      ARM+1    INCREMENTA
 87   E38 85               INC      *        SEGUNDA PALAVRA
 88   E39 2E 35            ARM      ARM+1    DO ENDERECO
 89   E3B 96               SV       1        SE NAO DEU CARRY,
 90   E3C 0E 23            PLA      LIMPA    VAI LER PROXIMA PALAVRA
 91   E3E 4E 34            CAR      ARM      SE DEU,
 92   E40 85               INC      *        INCREMENTA PRIMEIRA PALAVRA
 93   E41 2E 34            ARM      ARM      GUARDA
 94   E43 D1 0F            DD       4        TESTA SE DEU CARRY
 95   E45 D8 FE            SOMI     -/02     ALSM DO BIT 11
 96   E47 B5 23            PLAZ     LIMPA    NAO: VAI LER PROXIMA PAL.
 97   E49 86       UNEG    UNEG     *        SIM: PARA EM LOOP
 98   E4A 9D               PARE     *        COM /FF
 99   E4B 0E 49            PLA      UNEG     NO ACUMULADOR
100   *
101   *
102   * PROTECAO DO PROGRAMA PARA QUE NAO SEJA DESTRUIDO PELOS DADOS
103   *                        OU POR ENDERECAMENTO INVALIDO.
104   *
105   *
106   E4D DA 00    CARI    CARI     /00      RESTAURA ACUMULADOR
107   *
108   E4F 00 00    PROTEGE PLA      0        RETORNA
109   E51 2E 4E            ARM      CARI+1   SALVA ACUMULADOR
110   E53 4E 34            CAR      ARM      SEPARA 4 BITS
111   E55 D1 4F            DE       4        MAIS SIGNIFICATIVOS
112   E57 D1 6F            GE       4        DO ENDERECO
113   E59 D8 F2            SOMI     -/0E     TESTA SE ENDERECO >= /E00
114   E5B AE 4D            PLAN     CARI     NAO: VAI RETORNAR
115   E5D 0E 49            PLA      UNEG     SIM: VAI PARAR EM LOOP
116   *
117   * ENTRAS DRIVER DE ENTRADA DE DA.OS, SEM BIT DE PARIDADE
```

```
118  *
119  E5F 00 00  ENTRA   PLA     0
120  E61 CB 11          FNC     /B1      CLF
121  E63 CB 16          FNC     /B6      STC
122  E65 CB 21  WAIT    SAL     /B1      ESPERA
123  E67 0E 65          PLA     WAIT     FLAG
124  E69 CB 40          ENTR    /B0      ENTRA DADO COMPLEMENTADO
125  E6B 82             CMP1    *        DESCOMPLEMENTA
126  E6C D1 41          DE      1        LIMPA PARIDADE
127  E6E D1 21          CD      1        DO DADO
128  E70 0E 5F          PLA     ENTRA    RETORNA
129  *
130  * SAI - DRIVER DE SAIDA
131  *
132  E72 00 00  SAI     PLA     0
133  E74 CB 80          SAI     /B0      SAI DADO
134  E76 CB 21  WFF     SAL     /B1      ESPERA
135  E78 0E 76          PLA     WFF      FLAG
136  E7A 0E 72          PLA     SAI      RETORNA
137  *
138  * LECONV = ROTINA DE "CONVERSAO" HEXBIN
139  *
140  E7C 00 00  LECONV  PLA     0
141  E7E DA 0F  IGNOR   CAR!    /0F      FAZ INDICE
142  E80 9E             TR'     *        _/0F(NUMERO DE DIGITOS)
143  E81 FE 5F          PU;     ENTRA    OBTEM DADO
144  E83 83             CM'2    *        TROCA SINAL
145  E84 2E C4          ARM     ACC      SALVA DADO COMPLEMENTADO
146  E86 4E C4  LOOP    CAR     ACC      CARREGA DADO COMPLEMENTADO
147  E88 BE 7E          PLAZ    IGNOR    SE FFRM, IGNORA-O
148  E8A 7E C5          SOMX    DIGITOS  TESTA SE E' DIGITO
149  E8C BE 82          PLAZ    ACH      SE FOR, =) ACH
150  E8E E0 00          S 'S    0        SE NAO, APONTA PROXIMO
151  E90 0E 86          PLA     LOOP     E VAI TESTAR NOVAMENTE
152  E92 4E C4          CAR     ACC      SE NAO DIGITO, RECUPERA DADO
153  E94 D8 20          SOMI    @        TESTA SE E' BRANCO
154  E96 BE B5          PLAZ    BRANCO   SIM =) BRANCO
155  E98 4E C4          CAR     ACC      NAO=) CARREGA DADO
156  E9A D8 0D          SOMI    /0D      TESTA SE E' RETURN
157  E9C BE B5          PLAZ    BRANCO   SIM=) BRANCO
158  E9E 4E C4          CAR     ACC      NAO=) TESTA SE
159  EA0 D8 0A          SOMI    /0A      E' LINEFEED
```

```
160  EA2 BE 7E           PLAZ    IGNOR   SIM=> IGNORA
161  EA4 4E C4           CAR     ACC     NAO=> TESTA SE
162  EA6 D8 40           SOMI    @@      ARROBA
163  EA8 BE B8           PLAZ    ARROBA  SIM=> ARROBA
164  EAA DA 5F           CARI    @_      NAO! INVALIDO!
165  EAC FE 72           PUG     SAI     IMPRIME "_" NA CONSOLE
166  EAE 80              LIMPO   *       ZERA ACUMULADOR
167  EAF 9D              PARE    *       PARA
168  EB0 0E 7E           PLA     IGNOR   E IGNORA O CARATER
169  EB2 9E     ACH      TRI     *       (INDICE=DIGITO CONVERTIDO)
170  EB3 0E 7C           PLA     LECONV  JOGA NO ACC E VOLTA
171  EB5 86     BRANCO   UNEG    *       RETORNA C/ -1
172  EB6 0E 7C           PLA     LECONV  SE FOR BRANCO OU RETURN
173  *
174  EB8 DA 0D  ARROBA   CARI    /0D     SAI
175  EBA FE 72           PUG     SAI     RETURN
176  EBC DA 0A           CARI    /0A     SAI
177  EBE FE 72           PUG     SAI     DOIS
178  EC0 FE 72           PUG     SAI     LINEFEEDS
179  EC2 0E 01           PLA     LEENDER VOLTA A LER ENDERECO.
180  *
181  * BUFFERS E CONSTANTES
182  *
183  *
184  EC4 00     ACC      DEFC    0       P/ SALVAR ACUMULADOR
185  EC5 30     DIGITOS  DEFC    @0
186  EC6 31              DEFC    @1
187  EC7 32              DEFC    @2
188  EC8 33              DEFC    @3
189  EC9 34              DEFC    @4
190  ECA 35              DEFC    @5
191  ECB 36              DEFC    @6
192  ECC 37              DEFC    @7
193  ECD 38              DEFC    @8
194  ECE 39              DEFC    @9
195  ECF 41              DEFC    @A
196  ED0 42              DEFC    @B
197  ED1 43              DEFC    @C
198  ED2 44              DEFC    @D
199  ED3 45              DEFC    @E
200  ED4 46              DEFC    @F
████████████████████████████████ 201   000                  FIM     0
```

h) Exemplos de Programas Relocáveis:

```
CON  000  ENT
SUB  ***  EXT
ARF  ***  EXT
CAF  ***  EXT
SEN  ***  EXT
MAT  00E  ABS
PIS  012
ACF  00A  ABS
OFW  012  ABS
```

```
/00 SI
```

```
PASSO2
```

```
  1                      @BLT
  2  000                          SUBR     COSEN
  3                      * ROTINA QUE CALCULA O COSENO NO PATINHO
  4                      * PELA FORMULA COS(X)= SEN(PI/2 - X)
  5                      *
  6  000                          ENT      COSEN
  7  000                          EXT      SUB
  8  000                          EXT      ARMACF
  9  000                          EXT      CARACF
 10  000                          EXT      SEN
 11  000 00 00           COSEN    PLA      0
 12  002 F0 00 X                  PUG      ARMACF
 13  004 01                       DEFC     1
 14  005 00 0E                    DEFE     MANT
 15  007 F0 00 X                  PUG      CARACF
 16  009 01                       DEFC     1
 17  00A 00 12 R                  DEFE     PISDOIS
 18  00C F0 00 X                  PUG      SUB
 19  00E F0 00 X                  PUG      SEN
 20  010 00 00 R                  PLA      COSEN
 21  00A                 ACF      EQU      /00A
 22  00E                 MANT     EQU      /00E
 23  012                 OFLOW    EQU      /012
 24  012 64              PISDOIS  DEFC     /64
 25  013 87                       DEFC     /87
 26  014 D0                       DEFC     /D0
 27  015 01                       DEFC     /01
 28  000                          PLA
```

```
DIV  000  ENT
SAA  ***  EXT
NOM  ***  EXT
NAM  ***  EXT
TAB  ***  EXT
ARF  ***  EXT
SGL  ***  EXT
COM  ***  EXT
SOI  ***  EXT
CAF  ***  EXT
SHL  ***  EXT
TAC  ***  EXT
SHR  ***  EXT
PON  ***  EXT
RET  ***  EXT
OFW  012  ABS
ZEO  017  ABS
FOS  01A  ABS
ACF  00A  ABS
MAT  00E  ABS
DFT  01E  ABS
QUI  016
GOO  07B
SOS  05A
MOE  030
YES  045
GOL  06E

/00 SI

PASS02
```

```
 1                       OBLT
 2   000                          SUBR     DIV
 3                       *
 4                       *  DIV   -  ROTINA DE DIVISAO EM PONTO FLUTUANTE
 5                       *                     ACF = ACF/MANT
 6                       *
 7   000                          ENT      DIV
 8   000                          EXT      SALVA
 9   000                          EXT      NORM
10   000                          EXT      NADABEM
11   000                          EXT      TAB
12   000                          EXT      ARMACF
13   000                          EXT      SGNAL
14   000                          EXT      COMPLEM
15   000                          EXT      SOMATRI
16   000                          EXT      CARACF
17   000                          EXT      SHIFTL
18   000                          EXT      TAC
19   000                          EXT      SHIFTR
20   000                          EXT      POESIN
21   000                          EXT      REST
22   012                 OFLOW    EQU      /012
23   017                 ZERO     EQU      /017
24   01A                 F        EQU      /01A
25   00A                 ACF      EQU      /00A
26   00E                 MANT     EQU      /00E
27   01E                 OFLOAT   EQU      /01E
28                       *
29   000 00 00           DIV      PLA      0
30   002 F0 00 X                  PUG      SALVA    SALVA ACC,EXT,INDICE,T,V.
31   004 80                       LIMPO    ZERA-SE O
32   005 20 12                    ARM      OFLOW    INDICADOR DE OVERFLOW
33   007 F0 00 X                  PUG      NORM     NORMALIZA-SE ACF E MANT.
34   009 40 17                    CAR      ZERO
35   00B B0 16 R                  PLAZ     QUI      ACF#0 E MANT#0, VAI P/QUI
36   00D 08 FE                    SOMI     -2
37   00F 82                       CMP1
38   010 B0 7B R                  PLAZ     GO       SE ACF=0,VAI P/ GO.
39   012 F0 00 X                  PUG      NADABEM  SE MANT =0, VAI NADABEM.
40   014 00 7B R                  PLA      GO       E DEPOIS VAI P/ GO.
41   016 F0 00 X         QUI      PUG      TAB      TROCA ACF COM MANT.
42   018 F0 00 X                  PUG      ARMACF   ARMAZENA ACF
43   01A 01                       DEFC     1        EM F.
44   01B 00 1A                    PLA      F
45   01D 40 0D                    CAR      ACF+3    SUBTRAI-SE EXPOENTE DO DIVISOR
46   01F 83                       CMP2     DO       EXPOENTE DO DIVIDENDO
47   020 60 11                    SOM      MANT+3
48   022 90                       ST       0        SE HOUVE OVERFLOW,
49   023 F0 00 X                  PUG      NADABEM  VAI PARA NADABEM.
50   025 20 5B R                  ARM      SOS+1    SENAO, GUARDA RESULTADO.
51   027 F0 00 X                  PUG      SGNAL    FAZ PRODUTO DOS SINAIS.
52   029 F0 00 X                  PUG      COMPLEM  COMPLEMENTA ACF(DIVISOR).
53   02B F0 00 X                  PUG      TAB      DESTROCA ACF E MANT.
54   02D DA 17                    CARI     23
55   02F 9E                       TRI      LOOP     SERA FEITO 24 VEZES.
56   030 F0 00 X         MORE     PUG      ARMACF   SALVA ULTIMO RESTO
57   032 01                       DEFC     1        DAS SUBTRACOES SUCESSIVAS
58   033 00 1E                    PLA      OFLOAT   EM DFLOAT
59   035 81                       UM
60   036 99                       TRE      EXT      = 1
```

```
 61  037 F0 00 X          PUG    SOMATRI ACF = ACF - MANT
 62  039 40 0A            CAR    ACF
 63  03B 82               CMP1   SE      ACF >=0,
 64  03C A0 45 R          PLAN   YES     VAI PARA YES.
 65  03E 80               LIMPO  SE      ACF < 0, EXT= 0, EXT INDICA
 66  03F 99               TRE    SE      DIVISOR COUBE NO DIVIDENDO.
 67  040 F0 00 X          PUG    CARACF  CARREGA RESTO ANTERIOR
 68  042 01               DEFC   1
 69  043 00 1E            PLA    DFLOAT
 70  045 81         YES   UM     SHIFTA  ACF P/ ESQ. DE UM BIT.
 71  046 F0 00 X          PUG    SHIFTL
 72  048 F0 00 X          PUG    TAC     QUOCIENTE PARCIAL VEM P/ ACF
 73                 *                           E RESTO VAI DE ACF P/ CFLOAT
 74  04A 81               UM     SHIFTA  P/ ESQ. DE UM BIT,
 75  04B F0 00 X          PUG    SHIFTL  E
 76  04D 99               TRE    INTRODUZ NO ULTIMO BIT DA
 77  04E 60 0C            SOM    ACF+2   DIREITA, O BIT QUE ACABA
 78  050 20 0C            ARM    ACF+2   DE SER CALCULADO
 79  052 F0 00 X          PUG    TAC     QUOCIENTE PARCIAL VAI P/ CFLOAT
 80                 *                           E RESTO VOLTA P/ ACF.
 81  054 E0 00            SUS    0
 82  056 00 30 R          PLA    MORE    FIM DO LOOP.
 83  058 F0 00 X          PUG    TAC     CARREGA RESULTADO EM ACF.
 84  05A DA 00      SOS   CARI   0       SOS+1 CONTINHA DIFERENCA DOS
 85  05C 20 0D            ARM    ACF+3   EXPOENTES E ESTA VAI P/ACF+3.
 86  05E 40 0A            CAR    ACF
 87  060 82               CMP1   SE      PRIMEIRO BIT DE ACF FOR 1,
 88  061 A0 6E R          PLAN   GOL     TEMOS QUE
 89  063 81               UM     SHIFTAR P/DIR. UM BIT.
 90  064 F0 00 X          PUG    SHIFTR  E
 91  066 40 0D            CAR    ACF+3   INCREMENTAR EXPOENTE
 92  068 85               INC
 93  069 90               ST     0
 94  06A F0 00 X          PUG    NADABEM SE TRANSBORDOU, VAI P/ NADABEM.
 95  06C 20 0D            ARM    ACF+3
 96  06E F0 00 X    GOL   PUG    POESIN  COLOCA SINAL NO RESULTADO,
 97  070 F0 00 X          PUG    NORM    NORMALIZA,
 98  072 F0 00 X          PUG    TAB
 99  074 F0 00 X          PUG    CARACF  RESTAURA O DIVISOR
100  076 01               DEFC   1       EM MANT,
101  077 00 1A            PLA    F
102  079 F0 00 X          PUG    TAB
103  07B F0 00 X    GO    PUG    REST    RESTAURA ACC,EXT,INDICE,T,V,
104  07D 00 00 R          PLA    DIV     E PRONTO.
105  000                  FIM
```